UNIVERSITY OF CALIFORNIA
SANTA CRUZ

NATURAL LANGUAGE INTERFACES FOR PROCEDURAL CONTENT GENERATION IN GAMES

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Afshin Mobramaein Kano

March 2020

The Dissertation of Afshin Mobramaein Kano
is approved:

Professor Jim Whitehead, Chair

Professor Luca de Alfaro

Assistant Professor Adam Smith

Quentin Williams
Acting Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Natural Language Interfaces for Procedural Content Generation in Games

by

Afshin Mobramaein Kano

Mixed-Initiative Procedural Content Generation (MI-PCG) focuses on developing systems that allow users with diverse technical backgrounds to co-create interesting and novel game content in collaboration with a computational agent. These systems provide a front-end for users to interact with a generator by means of placing different constraints, or modifying a variety of the generator's parameters. While these systems provide significantly enhanced design support over traditional design tools, there exists areas of opportunity to address shortcomings in these systems such as high user interface complexity (too many controls presented, little feedback provided) and the lack of a model of designer intent (the system can reason over constraints but does not understand the expressive intent of the user).

We believe that natural language interfaces can provide a way of addressing these areas by utilizing the expressiveness of natural language as an input for mixed-initiative systems in a way that it can reduce interface complexity by converting natural language queries into design space movements or constraints for the generator to act upon. By reducing all input to a single query the natural language interface can make the appropriate selection of parameters and controls that can result in the desired result for the user compared to the traditional

modification of one control at a time in a traditional graphical user interface. Furthermore, the issue of designer intent can be addressed by creating a mapping of natural language concepts into a series of parameter combinations that allows for multi-dimensional movements in the design space of the generator, rather than manipulating a series of controls sequentially to achieve the same effect.

In this thesis we explore the design and implementations of natural languages in MI-PCG systems through the development of a design methodology for encoding natural language understanding into MI-PCG systems and the implementation of two proof of concept systems named CADI and WATER4-NL for different use case scenarios such as automated game design and shader manipulation respectively. Furthermore a user study based evaluation of WATER4-NL and its results are presented and discussed.

To my family, thank you for all the support you gave me along the way.

Acknowledgments

First of all, I would like to thank my parents Hamid Mobramaein and Florina Kano, as well as my brother Farid Mobramaein for the great amount of support that they provided me while pursuing my PhD. Without your help, all of this could not have happened. Thank you for the countless hours of support you gave to make this dissertation happen,and for helping me out in any way you could through the good and through the bad times in these years as a graduate student.

I would also like to thank my dissertation advisor Jim Whitehead. Thank you for your mentoring throughout the years, without all your help this dissertation would not have come to fruition. Thank you for believing in me even in my worst times, and for helping me navigate the twists and turns of the Ph.D program from the beginning up to this moment. In addition, I would also like to thank my committee members Adam Smith and Luca de Alfaro for providing their feedback and guidance throughout this process.

Finally I want to thank my collaborators and friends: Arjun Rao, Suzanne da Câmara, Emily Hendershot, Ryan Compton, Lisa Gatlin, Ross Greenwood, Andres Gonzalez, Diego Aguilar, Felipe Gonzalez, Morteza Behrooz, and Chandranil Chakraborttii for helping me make the research in this thesis happen. All the help writing the papers, testing my software, and providing feedback made this work here happen.

Chapter 1

Introduction

1.1 Mixed Initiative Design and Games

From the integration between man and machine envisioned in "Man-Computer Symbiosis" [28] to the sketch-based interactive design capabilities of the soft architecture machine [35], researchers have long sought tools that create a design collaboration between people and computers. Such systems today tend to be called either mixed-initiative, emphasizing the nature of turn-taking between human designers and computer designers, or co-creative, emphasizing the contributions of humans and computers without necessarily implying a turn-taking approach. This interaction modality suits itself well into the process of game design and development where diverse specialists such as level designers, artists, and programmers iteratively collaborate towards a common artifact, in this case, a game. In this

sense, mixed-initiative interactions within a game design context can be seen as a designer talking to a specialist (such as an artist or level designer) and iteratively working, or co-creating, a solution that requires the specialist's domain knowledge, in this case the specialist knowledge would be embedded into a computational agent. This is where Procedural Content Generation (PCG) comes into play: For years, the field of PCG has developed a series of computational agents that embed the knowledge of game design specialists and are capable of generate almost infinite amounts of content. PCG systems are capable of automating a wide variety of game design tasks such as level design, music composition, visual asset generation, and even complete games as in the case of Automated Game Design (AGD) systems.

Mixed initiative design systems have attracted considerable interest within the procedural content generation (PCG) for games community. Traditionally, procedural content generation within games has focused on creating game content—such as a game level or map—with little to no input from the player. The classic dungeon crawler Rogue exemplifies this approach, with each level of a dungeon being generated by a computer algorithm, with no input from the player [59]. Designer aesthetics are embedded in the generation algorithm. The player either accepts a generated dungeon and plays on, or rejects it by ending their game session. In contrast, a mixed-initiative procedural generation system creates a collaboration between a human designer and a procedural generation algorithm,

similar to the analogy between the designer and the specialist mentioned above.

The Tanagra [53] system demonstrates mixed initiative design in the arena of level design for 2D platformer games similar to Super Mario Bros [29]. The human designer places one or more platforms, thereby creating a partial level design. Tanagra's generator reacts to these platform placements, and automatically generates a suggested design for the remainder of the level. This suggested design can then be modified by the designer, leading to further design suggestions, and so on. Other recent examples include Sentient Sketchbook [27], a strategy map design tool, Casual Creators [12], and mixed-initiative game design tools for mobile devices [36, 37].

1.2 Why talk when you can just push a button?

Current generation mixed-initiative design tools for games provide significantly enhanced design support over traditional tools that provide a blank canvas to human designers. However, there are several ways one might ideally like to improve these systems. First, existing tools constrain and channelize design activity via their user interface affordances. For example, in Tanagra [53] UI only permits the manipulation of platforms and placement of non-player characters, thus limiting design activity to these facets of gameplay. Second, existing tools don't have a rich model of designer intent, and this limits the kinds of design assistance they can provide. Tanagra's model of designer intent is limited to the platforms placed by

the designer, and the notion of "pinning" a platform to a fixed location. Whether the human designer is creating a fast-paced hard level, or a slow-paced easy level is beyond Tanagra's understanding. Finally, lacking a model of intent, it's not possible to manipulate designer intent over time. It isn't possible to ask Tanagra to make a level "more frantic" or to interpret suggestive but ambiguous desires like "make it colder".

Another challenge with traditional mixed initiative design tools for games is their interface complexity. Designers working with these tools explore a high dimensional design space. An example of this is Cillr, a mixed-initiative game creation system [36, 37]. This system has an interface with 284 controls, one for each feature of their knowledge representation for games. Nelson et al. [37] mention challenges during user testing relating to difficulty navigating the UI and understanding of the design space that stem from the high dimensionality of the design space, and the complexity of the user interface in the system.

This presents an opportunity to explore different interfaces in the creation of mixed-initiative PCG systems. Natural language interfaces in this case can provide an alternative to GUIs with a large amount of fixed controls presented at once to the human designer. The dialogue-based paradigm of mixed-initiative design is well suited to the turn-based interaction of conversational natural language interfaces. Human designers can take advantage of the conversational nature of these interfaces to explore the design space of an artifact by moving one charac-

teristic at a time in an incremental fashion until they reach their objective. This step-by-step design space exploration combined with a real-time visualization of the generated artifact as it changes throughout the design process can provide an alternative to the complex UIs that mixed-initiative systems possess.

Natural language interfaces can be used in different case scenarios across the game design and development process. For example, one use scenario of natural language interfaces in mixed-initiative design system is the one of co-creative game design. Games as a finished artifact are generally described by human designers and users in qualitative terms, rather than quantitative. One can think of describing a video game as "frantic", "smooth", or "stressful" but rarely one describes games in numerical quantities and parameters. As such, using mainly quantitative values while exploring the space of generated games in a mixed-initiative tool might frustrate the human designer during the process. On the other hands, iteratively exploring the design space by describing what aspect of the game is being explored at a time might prove more useful to the human designer. One could think of modifying a parameter of a game by saying "Make the character move faster" feel more appropriate as a descriptive characteristic of a game in its design process rather than quantitative descriptions like *"character.xSpeed = 32"*. The former type of interactions in the design process of games lends itself as an opportunity to explore the usage of natural language interfaces in mixed-initiative game design.

1.3 When talking about it is not enough.

While the usage of conversation-based interfaces might be able to address some of the UI design issues of mixed-initiative game design systems, there are issues to be considered when implementing such a system. One of the issues of natural language interfaces in mixed-initiative systems is how much can a designer interact with a conversational based interface continuously before finding the experience frustrating. This is an analogue to the problem of interface complexity in control-based UIs. While the large amount of controls presented to the user might prove frustrating and hard to navigate to the human designer, an extended interaction with a natural language interface might frustrate the user. This could be interpreted by the designers as the system "not listening" to their input if the results of their conversations about a design do not result in their expected vision of the artifact.

A second issue is the one of finding a starting point between the system and a human designer such that the exploration of the design space of our system leads to a successful co-creation process. This "blank-canvas" process carries several design considerations such as whether either a random solution or a fixed initial point of entry affects the exploration of the design space of our artifacts. In addition, given the conversational nature of the interface the proposition of who initiates the co-creation process arises. Should the designer initiate the exploration of the design space by selecting the parameter they feel is the most appropriate

to modify to realize their vision? Or should the system act as a guide by pointing at parameters that might be able to achieve the designers vision in an efficient manner? This is an interesting consideration, since a designer-initiated process might lead to an efficient pruning of the design space of the system, since the user is expected to direct its vision towards the system. On the other hand, a system initiated co-creation process might lead the designer to consider parts of the design space of the system that otherwise would be ignored by letting the system lead the process.

This leads us to the issue of design workflows while using natural language interfaces. One feature that is present in graphical UIs in mixed-initiative systems is the freestyle workflow that having all options presented at once affords the designer. In this sense, a more linear workflow is present in a conversational metaphor. By iterating one aspect of the design at a time in an ordered manner, the human designer might become frustrated by the system. For example, the designer might perceive that they have to methodically go through a phone-tree style menu to reach the aspects they desire to modify. This can become a cumbersome task in the designer's mind as they feel they cannot apply their workflow to a turn-based interaction model. In this sense the system's conversational interface needs to present the affordance of being "freestyle" by letting the designer move around the design space freely in any order. These above are some of the issues that can arise in the design of natural language interfaces for mixed-initiative

co-creative systems. As such, the designer needs to consider these possibilities in order to embrace the advantages that this metaphor affords.

1.4 Research Questions

In this thesis we explore the design and implementation of natural language interfaces for MI-PCG systems by answering the following research questions:

- How do we convert natural language queries into actionable affordances within a mixed-initiative interface?
- How do we encode actions and designer intent in a natural language interface?
- How do we design natural language queries that capture the affordances of the system?
- How can we implement such an interface and what use-case scenarios can benefit from it?
- How do we evaluate the effectiveness of a natural language interface in mixed-initiative procedural content generation?

1.5 Contributions

This dissertation introduces contributions to the fields of mixed-initiative interface design, procedural content generation, and natural language interface design in the form of the following works:

- A design methodology for natural language interactions and designer intent modeling in MI-PCG systems.
- Two natural language interface based MI-PCG systems: CADI and WATER4-NL designed for different use-case scenarios (Automated Game Design and Shader Manipulation)
- A user-study based evaluation comparing natural language input versus traditional graphic user interfaces in procedural content generation systems using the WATER4-NL system.

The following chapters will cover a literature review of the research that has informed the development of the work in this thesis and a discussion of each of the major contributions presented in this work.

Chapter 2

Related Work

2.1 Introduction

In this chapter, we cover the works that have informed the development of this dissertation, such as mixed-initiative interfaces, procedural content generation, automated game design, and natural language interfaces. This literature survey covers the core concepts upon which we build the works presented in this thesis. We first introduce the concept of procedural content generation (PCG), the major field upon which this work is contributing. Then we focus on two types of PCG systems: mixed-initiative and automated game design. Finally, we look at the development of natural language interfaces outside the field of procedural content generation.

2.2 Intro To Procedural Content Generation

Procedural content generation (PCG) is defined as the "algorithmic creation of game content with limited or indirect user input" [51]. We interpret this as using programmatic techniques (algorithms) to generate different types of content, such as levels, rules, visual assets (textures, models), and music. One of the principal motivations for implementing PCG systems is that algorithmic techniques can allow users to create large amounts of content while reducing the development overhead created by manually authoring game content.

Besides, the implementation of a PCG system in a game can also benefit players too. Having a PCG system that generates a large amount of game content can increase the replay value of a game compared to a traditionally hand-crafted game system where there is a limited amount of content. One example of this is the usage of a level generator in a game. This approach to game development can save the designers and developers time by creating a large number of levels consistent with the game's design specifications. PCG also provides the players a more substantial amount of gameplay time since the number of levels to play is significantly larger than when using an authored approach. On the other hand, the academic community has focused on providing game designers and developers with algorithms and tools that use state of the art artificial intelligence techniques to create high-quality game content algorithmically.

2.3 Mixed-Initiative PCG

Another way of looking at the why's of PCG system implementation is from a design support perspective. Designing game-agnostic PCG tools such as music, text, and asset (model) generators can help lower the technical barrier of entry to aspiring designers to express their vision when it comes to game development. MI-PCG systems are of particular note here, as they focus on tools that can facilitate and semi-automate specific design tasks like level generation, music composition, visual asset creation, up to game concept prototyping.

One key difference between "traditional" PCG systems and MI-PCG system is that in MI systems, the designer interacts iteratively with a computational agent (algorithmic designer) until the desired generated content complies with the designer's constraints. In comparison, in a "traditional" system, the designer either provides a set of initial parameters, or even no input, to the PCG system and "trusts" that the output of the system complies with their desired constraints. By adding a human-in-the-loop to the procedural content generation process, we can create systems and tools that allow users to create game content without having to learn the innards of the procedural generator behind it.

2.3.1 Interaction Modalities

Liapis, Smith, and Shaker [51] mention two different types of interaction modalities in mixed-initiative PCG relative to the amount of autonomy the pro-

cedural generation back end has. On one side, there are Computer-Aided-Design (CAD) systems. In this type of system, the procedural generator only performs when the user passes the necessary constraints to the algorithms in the back-end. This type of system allows the user to interact iteratively with the algorithmic back-end while still obscuring the internals of the procedural generation process to the user. This adds an interactive layer of abstraction to traditionally autonomous procedural content generation systems. Analogously, the user, in this case, takes the role of an art director telling the artist (in this case, an algorithm) how to create a piece that meets their standards. There exist several mixed-initiative systems of this type.

Some examples of CAD mixed-initiative systems are: Tanagra [53], a mixed-initiative level design tool for creating 2D platformer game levels, Sentient Sketchbook: a tool for co-creating strategy game levels, Cillr and Wevva [36, 37, 45] two systems for creating game prototypes on mobile-devices, and a system for creating level progressions for the game Refraction. [42, 6].

On the other hand, there are the interactive evolution systems. If CAD systems are analogous to the work of an art director, interactive evolution systems are similar to buying a new pair of glasses. In this type of interaction modality, the system outputs potential solutions first, and the user selects the ones they prefer to reduce the search space of possible solutions. These systems, while still iterative in their interaction nature, do not wait for the user to act to start searching the

design space of the generator. Systems that employ this modality have been used for different use-cases such as: Interactive art generation with Picbreeder [48], strategy level generation [26], racing track generation [7], and mobile game rule creation [20].

In this thesis, we focus on building systems that employ the CAD interaction modality. Natural language input allows for the creation of rich, expressive constraints that translate into movements in the design space of the generator behind it. These natural language expressions are well suited to the "conversational" aspect of CAD systems. While in traditional CAD systems, the user has a conversation with the generator by manipulating controls. We propose that natural language interfaces allow the user to have a literal "conversation" with the generator about the content they desire to create. By using translating natural language into control manipulations, we create a new abstraction layer that permits the user to "talk" to the generator without having to think about any internal parameters.

2.3.2 Casual Creators

Casual creators [12] are one type of mixed-initiative system characterized by their goals of providing a simple, intuitive, and entertaining user experience. Rather than focusing on exposing the user to a wide array of capabilities and functionalities, as seen traditionally in design tools, casual creators focus on providing

the user with an experience that requires little to no prior technical knowledge about the domain, have simple easy to use interfaces, and provide as much assistance to the user as possible without taking autonomy away from them. Compton and Mateas [12] express these characteristics as a series of design patterns. These design patterns serve as a set of best practice guidelines to implement simple yet highly expressive design tools. Out of all the patterns we are particularly interested in the following:

Instant Feedback

We want to provide the users with instantaneous feedback about the operations they perform with the natural language. This is important because not presenting consistent feedback about the artifact the users are modifying can cause interaction fatigue and frustrate the user. This is especially important with natural language as the commands are more expressive than a simple value declaration, and thus we need to provide the user with consistent instantaneous feedback to either guide them in the direction of their goals, or to correct for mistakes they might make along the way.

No Blank Canvas

Casual creators focus on not providing a blank canvas solution (for example, an empty image) to the user, as this might cause more frustration. Given the iterative nature of mixed-initiative interactions and the high expressiveness of

natural language, figuring out an initial reference point in the design space might be a cumbersome task for the users. As such, we want to follow this design pattern to provide effective interactions with our natural language interfaces.

Limiting Actions to Encourage Exploration

By reducing input to only natural language input, contrasted to traditional generators that have all parameters exposed at once, we encourage the user to explore the design space in a more guided manner. By allowing them to modify the artifact one command at a time, the user feels encouraged to explore around by choosing aspects in descriptive terms rather than having them modify parameters one by one, which can make the users pigeon-hole themselves into particular sub-regions of the design space.

Modifying The Meaningful

Casual creators are designed in such a way that the reduced set of actions they can perform with the generator produce significant, but meaningful movements in the design space. Rather than expose the user to a high level of granularity by showing them every possible modifiable parameter at once, we use combinations of parameters mapped to a natural language concept. This way, the user is modifying a more considerable amount of features in the design space, rather than focusing on very granular value modifications. By providing more significant movements in the design space, we allow the user to think in bigger picture terms about the

artifact instead of focusing on minor details.

We believe that by following these design patterns, we can create meaningful natural language interactions that simplify the procedural content generation process by allowing the user to think at a higher level about the artifact, without having to learn the technical details behind its generation.

2.4 Automated Game Design

Automated Game Design (AGD) is a sub-field of procedural content generation that focuses on the process of creating complete games algorithmically. AGD systems use different types of AI techniques to generate complete playable experiences autonomously. For example, the first AGD system, METAGAME [43], builds variations of chess games based on previously authored knowledge about chess. Other systems such as Variations Forever [52] uses Answer Set Programming to generate game rule sets based on a series of logical declarations. Variations Forever uses constraint satisfaction techniques to reduce the search space and find tractable playable games. Finally, other systems use a generate-and-test approach to create new games. These systems use different search-based techniques to find solutions that are tractable and novel. Some example AGD systems of this kind are Togelius and Schmidhuber's "An Experiment in Automatic Game Design" [58], Cook's ANGELINA series[13, 14], and Browne and Maire's LUDI [5].

AGD systems generally operate over a more complex domain knowledge than

traditional procedural content generators. This complexity arises because traditional content generation techniques specialize in one type of content, such as levels, music, or graphical assets. In contrast, AGD systems have to deal with complex design spaces as they have to account for the many aspects that make a game, such as Interactions between agents, physics, level layouts, winning conditions, among many others.

2.4.1 Multi-Modal Parametric Design Spaces

As AGD systems deal with a complex domain knowledge such as games, knowledge representation is crucial to the operation of these systems. One of these representations comes in the form of parametric design spaces. In this type of knowledge representation, games are encoded as parameter-vectors in a design space. With this representation, every parameter combination represents a potential game in a discrete (albeit extremely large) possibility space defined by the designers. In that way, by encoding a complex artifact, like a game, into a vector, we can easily explore the possibility space by moving in it with simple numerical operations. These representations allow for a rich exploration of variations of a specific artifact without having to resort to computationally expensive search techniques.

For example, Isaksen et al. [24] utilized a combination of unsupervised learning techniques and evolutionary computation to explore a parametric design space of

variations of the game Flappy Bird[41]. By utilizing a vectorized representation, Isaksen et al. were able to utilize standard unsupervised learning techniques to find a set of variations of games that both covers a large amount of the design space, and is separated enough as the clustering optimized by genetic optimization maximizing centroid distances.

Another application of using a parametric design space in AGD is Cillr and Wevva [36, 37, 45] two mixed-initiative AGD systems that are capable of creating game prototypes on mobile devices. In these two systems, the user is allowed to explore a 284-dimensional design space represented in vectorial form. This design space covers a wide range of aspects of game design, such as physics, interactions, graphics, audio, level design, and game logic. The design of this parameter is based on the notion of fluidic games and expressed as a set of common characteristics of a set of game genres that can be mapped one to one to in the parameter space. This representation means that the games in this design space have already been created and are easy to explore by manipulation of the parameter. This allows for the creation of a simple, intuitive way for users to create games based on the parameters in the design space.

In this sense, we draw inspiration from the parameter space representation that allows moving in the design space by simple value manipulation (translated to GUI elements in the systems' implementation) which combined with a mapping of natural language concepts to one or more parameter allows for a simple compelling

exploration of a complex generative design space.

2.4.2 The Intersection Between AGD and Mixed-Initiative

Cillr and Wevva [36, 37, 45] are examples of how mixed-initiative interactions are applied in the domain of AGD. These two systems show how complex artifacts can be generated with ease, as these systems allow to create a full game prototype without having to write a single line of code or draw a single image for artwork. This ease of use keeps in line with the goals of casual creators as they significantly lower the technical barrier of entry, provide engaging feedback, and do not start with a blank canvas. Having interaction limited to only iteratively manipulating simple GUI controls (like sliders and knobs) shows how mixed-initiative interfaces, namely casual creators, can allow for engaging procedural content generation of complex multimodal output without exposing the internals of a complex procedural generation process.

The work made for Cillr and Wevva informs our implementations presented in the following chapters of this dissertation. These systems serve as an example of how to create simple interfaces that allow users to manipulate highly complex generative process with ease.

2.4.3 Natural Language and AGD

Natural language inputs and knowledge representations have been explored in the field of AGD. Some examples of these systems are Treanor et al. 's Game-O-Matic [60], which focuses on the generation of news games. News games are an abstraction of a news story in playable form. By taking a conceptual map of the agents in the story and the relationships between them, these links are complemented by a verb. The system constructs a game based on a combination of pre-authored mechanics (extracted from the verb links) that show the relationship between the extracted entities. Another system that takes natural language input is ANGELINA-3 by Cook, Colton, and Pease. [15]. ANGELINA-3 generates aesthetic game content such as graphics assets from a keyword theme. The system mines the web for audiovisual content that matches the theme keyword. These systems inform our work on how natural language concepts can translate into actions that a generator can take within a generative design space.

2.5 Natural Language Interfaces

Natural language interfaces have been utilized in a wide variety of professional domains. Some examples include using language input to navigate databases [19], programming in Java [46], visual analytics [49], and UAV mission planning [8]. One common motivation to implement natural language interfaces in these dif-

ferent domains stems from the desire to simplify complex technical tasks, thus making them accessible to more users. For instance Price [46] developed NaturalJava as a way of simplifying programming, by allowing novice programmers to describe simple code blocks as natural language commands. This way, the programming language's syntax is obscured to the user who in turn can think about the problem (programming) in a higher level of abstraction without having to worry about the syntactic details of their program. On the other hand, Chandarana et al. [8] mention their motivation to develop a natural language interface as stemming out of the need to reduce the complexity of the interfaces UAV operators have to use for mission planning. By using natural language, they can reduce the number of technical information they need to input into the mission planning system's interface and instead use natural communication methods to create a mission plan.

These motivations are analogous to what we present in this dissertation. By integrating natural language input into the procedural generation process, we want to both simplify a complex technical task, but also reduce the level of complexity of the generator's interface. We believe that the higher levels of abstraction that natural language input provides can reduce the level of difficulty that a novice user has to encounter when using a procedural content generation system. In this case, by allowing the user to describe their desired artifact in descriptive terms we obscure the technical components of the procedural generator. This helps the

user achieve their task in an efficient manner by eliminating the need to learn the technical details behind the generation process. While these details are certainly important, the user doesn't need to fully understand them to successfully interact with the system through natural language input. Furthermore, the reduction in interface complexity created by the use of natural language input allows for a reduction of the information the user needs to know when interacting with a tool.

2.5.1 System Initiative in Natural Language Interfaces

Natural language interfaces, due to the conversational nature of their interaction modalities can exhibit different levels of system initiative in an analogue manner to our examples from the mixed-initiative interaction modalities section in this chapter. One such example of how different levels of system initiative are encapsulated in natural language interaction can be seen in the domain of virtual assistants. Software such as Alexa[21] by Amazon and Siri [22] by Apple utilize natural language input to handle a variety of tasks such as making phone calls, searching for web results, calendar scheduling, messaging, and reminders.

In this type of system, the amount of system initiative is related to the clarity and amount of information the user inputs in their commands via natural language. If a clear and highly informative query is given, the system will respond appropriately without any further confirmation for the user. One such example is a query like "set an alarm at 8 in the morning". In this case the system recognizes

the query as "set alarm" for the action to be taken, and "at 8 in the morning" as the time to set the alarm. On the other hand giving it a query that is more ambiguous such as "set an alarm" will result in the system asking the user for extra information, in this case, the time to set the alarm. Using this example, the former query is analogous to a CAD system in which the generator responds to a user given constraint without further intervention. Whereas the latter is more in-line with the interactive evolution paradigm since the system presents a series of solutions that might better fit their intent. Figure 2.1 shows two use cases (setting an alarm and making a phone call) in which the amount of information in the query results in different levels of system initiative to achieve the same action using Siri on an Apple iPhone.

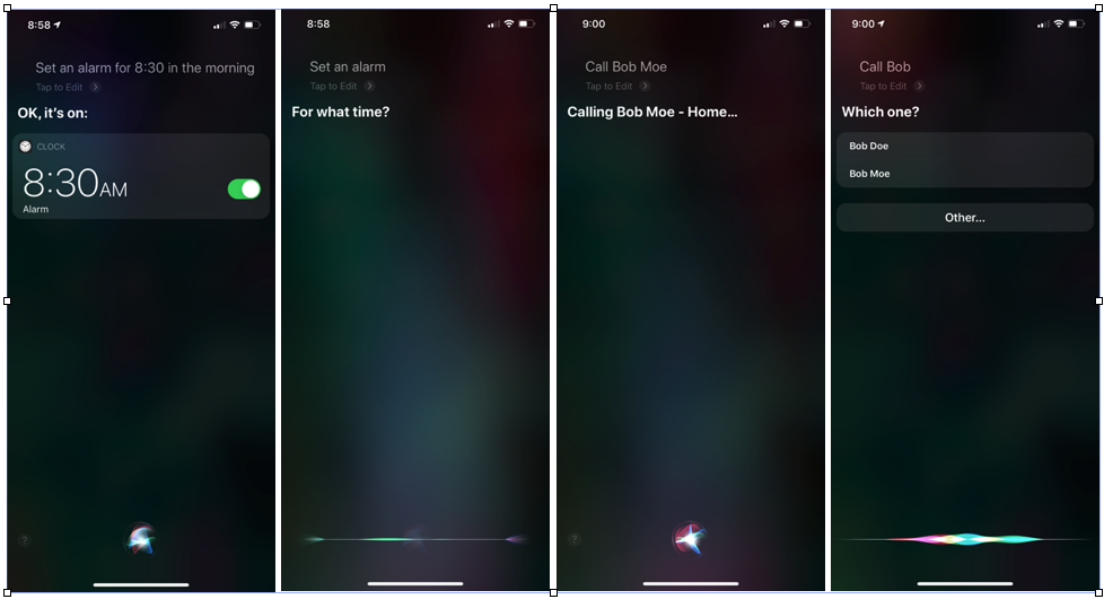


Figure 2.1: Two use cases that exhibit different amounts of user initiative based on the amount of information provided in their natural language query

Based on this analogy we can apply the same techniques used in other natural

language interfaces, such as the one exemplified above to the domain of mixed-initiative procedural content generation. In this sense, we can design specific query patterns that result in concrete constraints or design space movements for systems that are reactive to usere input, but we can use more ambiguous queries in systems that employ a large degree of initiative similar to interactive evolution systems as the user needs to iteratively collaborate with the system to achieve their goals.

2.6 Conclusions

In this chapter we have provided an overview of the research that influenced the works shown in this dissertation. The following chapters cover the implementation of two natural language interfaces called CADI and WATER4-NL. CADI is a natural language interface that explores the parametric space of Pong variations, this chapter is based off the publication in [30]. WATER4-NL is a natural language interface for the Water4 [56] plugin for the Unity3D [57] engine. The WATER4-NL chapter covers a methodological approach on how to build a natural language interface for an arbitrary procedural content generation system. The work in this chapter is based on the publication in [31]. Afterwards, we provide an evaluation of WATER4-NL in the form of a user study.

Chapter 3

CADI - A Conversational Assistive Design Interface For Discovering Pong Variants

3.1 Introduction

In the beginning of this thesis we made the case that natural language interfaces for mixed-initiative PCG systems can provide previously unexplored modes of creative support.

Natural language input (afforded via text or voice) allows the exploration of a design space using qualitative and affective terms. One example of this kind of interaction is a designer requesting a “zen game” or an “aggressive ball” rather

than manipulating a series of sliders in a traditional GUI based tool. These kinds of interactions might allow a designer to explore the design space of the tool in a way that understands their intent for the artifacts they look to create. In addition, exploration of the design space using qualitative and affective terms provides an interaction model that suits an operationalization of game design theories such as Game Feel [54].

Game Feel emphasizes the importance of affective experience in game design and focuses on controls, game world design, and aesthetic polish. As such, we can map affective natural language qualifiers such as cold, frenzied, and vibrant as a series of compound features in a parametric design space inspired by Colton et al.'s [11] concept of emergent features defined as "combinations of features that produce novel emergent effects". One could imagine an affective qualifier as a emergent-like compound feature that combines parameters that modify several aspects of the game at once. For example, a compound feature for "cold" could be a blue-hued color palette, slower moving character speeds, and white colored particle effects. This mapping of affective qualifiers to compound features in mixed-initiative tools could provide us with an operationalization of Game Feel that could be applied towards computational models of affective experience in PCG.

Exploring a parametric design space using affective qualifiers raises a series of research questions to be answered. How do we provide meaningful feedback to the user as they explore the design space of our mixed-initiative tool? Nelson et

al. [37] mention the occasional cases of user frustration while using Cillr when they modify a parameter slider in the tool, which results in an apparent lack of change in the artifact. We believe that by using compound features tied to a wide combination of game parameters spanning across graphics, physics, and sound, we can provide the user with meaningful feedback on where in the design space they find themselves in, and then take decisions that can lead them to their expected resultant artifact. This can be achieved by applying transformations to the parameters and visualizing the resultant artifact in real time while they interact with the natural language interface.

Another research question concerns the effectiveness of matching natural language input with designer intent. This question applies to understanding the quantitative nature of moves within parameter space to provide a meaningful exploration of the design space, as well as providing the user with a series of mappings of affective qualifiers that are able to cover a wide range of words. The question of understanding the quantitative nature of navigating the design space is important, since commands such as "make the game more intense" or "make the ball less quirky" can be interpreted in a wide variety of ways. With that in mind, it is important to develop an initial strategy of how granular the movement in the design space is, and also a mechanism to adjust the granularity of the movement according to user intent.

Finally, the question of how to handle the ambiguity of natural language input

for qualitative terms needs to be addressed, such user intentions can be understood as accurately as possible. For this, a classification of terms mapped to a series of compound features and parameters in the design space needs to be authored. This affords a flexible approach for grouping terms with similar features that can represent a similar concept. For example, terms such as warm, hot, and heated can be mapped to a feature controlling the palette of our game into a series of values with a red hue in the HSV color space, and to particle systems with a palette in the same tonality. Such a representation might help us understand basic models of polish for PCG as well as to provide an alternative to the parameter-value exploration of a design space.

In this chapter we introduce CADI (Conversational Assistive Design Interface), a natural language interface based PCG system that allows for the creation of variants of the game Pong [1]. This system is the first experiment in the development of natural language interfaces for mixed-initiative PCG explored in this thesis.

3.2 Related Work

Several works in the fields of automated game design (AGD) and mixed-initiative (MI) PCG informed the design and implementation of CADI.

Within the field of AGD, systems like Nelson and Mateas' [40] Interactive Game-Design Assistant and Game-O-Matic by Treanor et al. [60] are examples

of how a natural language representation of a concept can be used to generate Warioware Inc. [55] style micro games. In addition, systems such as ANGELINA-3 by Cook, Colton, and Pease [15] utilize natural language input to extract a context over which to apply a transformation over the aesthetic design space of a generated game. These systems provide examples of how natural language can be used to create the context of a generated game in the case of Game-O-Matic and Interactive Game-Design Assistant and how design space movements can be translated from a natural language input like in ANGELINA-3.

In the field of MI-PCG there are several example systems that showcase the interaction between a designer and a computational agent to create novel game content artifacts. For example, systems like Sentient Sketchbook [27], in this case a sketch, for the computational agent behind it to create a meaningful artifact that reflects the constraints of the sketch. As such, we model our part of our co-creation (collaborative mode) loop inspired on the interaction loop of this system. In addition, the UI developed in Sentient Sketchbook provided guidelines for CADI's implementation.

In addition, systems such as Ropossum [50] and Evolutionary Dungeon Designer [2] provide examples of how evolutionary computation can be used in mixed-initiative design for puzzle and dungeon level design by providing design assistance for playability and providing new solutions based on user constraints. At the intersection of AGD and MI-PCG, there are systems such as Cillr and Wevva [36, 37]

that provide tools that allow designers to co-create mobile mini games on-device. These tools provide an exploration of a fluidic game design space by allowing users to modify parameters in the design space and using an AI agent to evaluate the solutions they explore while using the system.

Outside of the domain of games and PCG, there are systems that enable collaboration through natural language between a computational agent and a human user. For example, Firedrop.ai's Sacha [61] is a chatbot based system that allows the creation of responsive websites from natural language input. In the field of human-robot interaction (HRI) there are systems that utilize spoken dialogue for collaboration between a mobile robot and humans [25] similar to the voice input modalities embedded into CADI that allow collaboration between a designer and our system through spoken inputs.

3.3 Expected User Interaction Modalities

In the design of CADI, we envisioned two different interaction modalities on how the user can explore the fluidic game design space for Pong variations. These two modalities were named collaborative and autonomous modes. In both modalities, the back-end design space exploration and game generation aspects are the same. The crucial difference rests on the amount of decisions the user takes while exploring the design space of Pong variants and its reflection in the user interface for CADI. This section explores both modalities in the system and provides an

explanation of the main interaction loop for our system. Figure 3.1 shows the interaction loop for CADI.

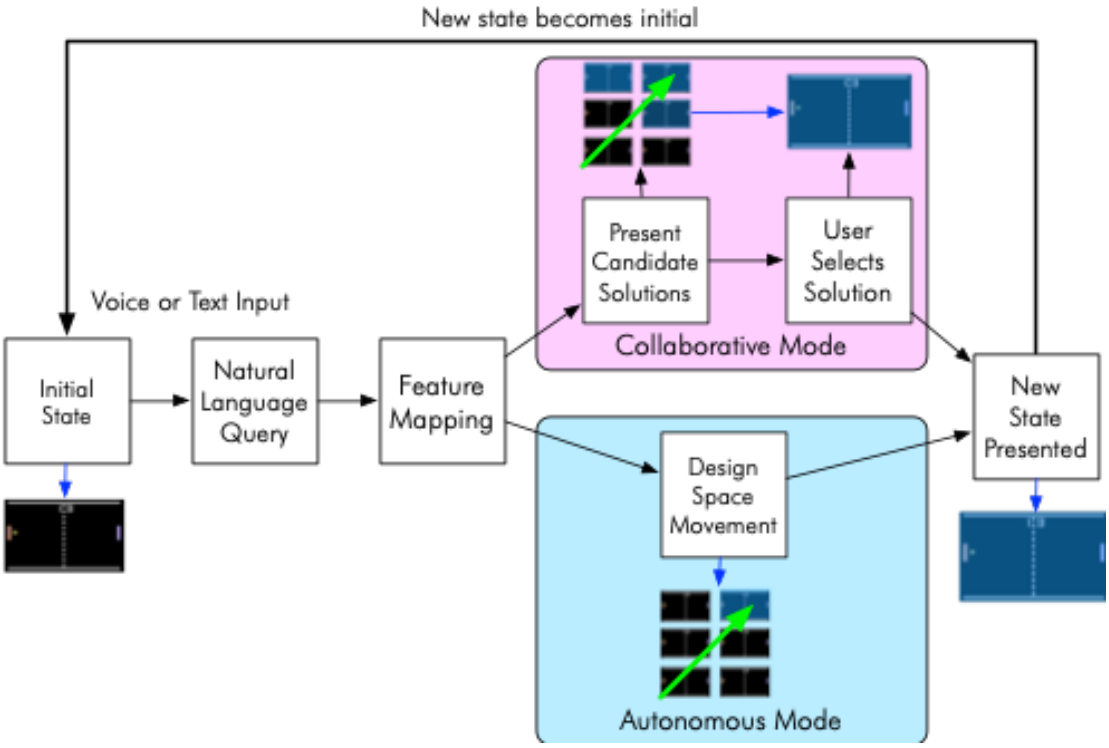


Figure 3.1: The interaction loop diagram for CADI showcasing two different modalities.

Our interaction loop in the system is comprised by the following steps:

1. The user is provided with an initial base game within our design space. This can be either a vanilla version of Pong faithful to its original design, or a random point in the design space. The user can also select one of these initial designs by using a button in the UI.

2. After being presented with a first point in the design space as a starter, the user can start exploring the design space by using natural language queries

either by voice input, or by text input. Some example queries are “make the game more aggressive”, “make the paddles heavier”, “make the ball red”.

3. The queries are passed to our natural language understanding back end, and the quantifier (i.e. less, more, way, quite...), agents (i.e. game, ball, paddle), and qualifier (i.e., vibrant, bouncy, warm...) are extracted. These are stored as a JSON object to be passed to our game generation system.

4. The game generation system receives the JSONified query and chooses the features mapped (i.e. vibrant = palettesaturation,nparticles) to the qualifier, determines the direction in which the design space is explored (positive, negative) from the quantifier and moves in the design space on the selected features.

5. The new design variation is presented to the user in a window inside the system's UI.

6. The user continues to make queries until finding a solution suitable to their original intent.

While the interaction loop is essentially the same for both autonomous and collaborative mode, the game generation step differs slightly amongst them. In autonomous mode the system moves in the design space and chooses one solution only for the user and presents it. In collaborative mode the system moves in the design space and chooses a series of neighboring points to the current solution by

applying a small amount of noise to a randomly chosen sub feature that composes the compound feature that is being modified and presents them to the user in order to accept the design change. This extra step provides the designer with multiple choices among different aesthetic variations of the same design.

3.4 System Architecture and Implementation

There are seven different components that make up the architecture of CADI. We implemented five different components to handle the pipeline for our mixed-initiative PCG system for exploring the design space of Pong [1] through a natural language interface. In addition, two components were provided by using operating system features (speech recognition/dictation), and a RESTful API (colourlovers.com) [33] that provides a color palette generation using the colourlovers.com API from a natural language tag. Figure 3.2 illustrates the pipeline architecture for CADI.

CADI was implemented by using the Unity3D engine for the UI, feature selection, game generation and rendering, while, Chatscript [62] handles the natural language understanding of queries. While designing and implementing CADI, several design decisions were taken in order to address the research questions and concerns introduced in at the beginning of this chapter:

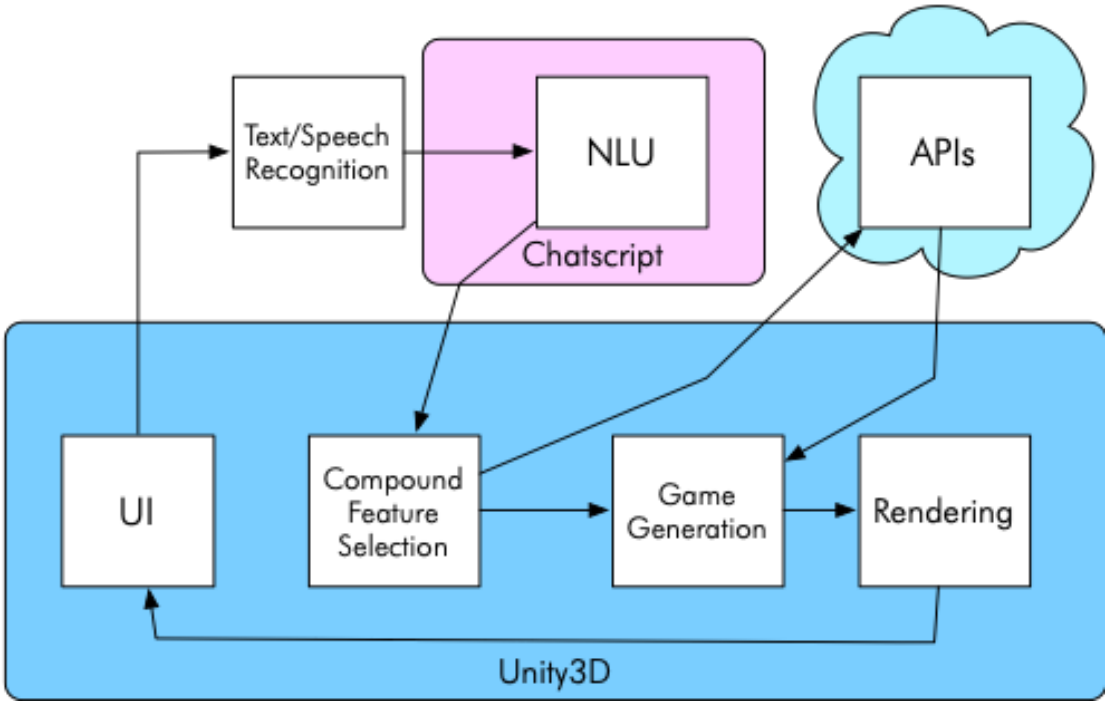


Figure 3.2: The pipeline architecture used in the implementation of CADI

3.4.1 Parameter Space Design

The first design decision addressed in the implementation of CADI was the one of choosing a design space for a natural language based mixed-initiative system. We informed our decision based on the work of Mobramaein et al. [32] about using Pong variations for exploring conversational interfaces in mixed-initiative PCG systems. This justification comes from the small number of agents in the game (ball, paddles, wall), simple collision based interactions, and high expressivity of the design space. This high expressivity can be seen in game variations such as Video Olympics [17] which uses the elements of Pong to create mechanical variations that reflect different sports, and games such as Pongs [3] that explore

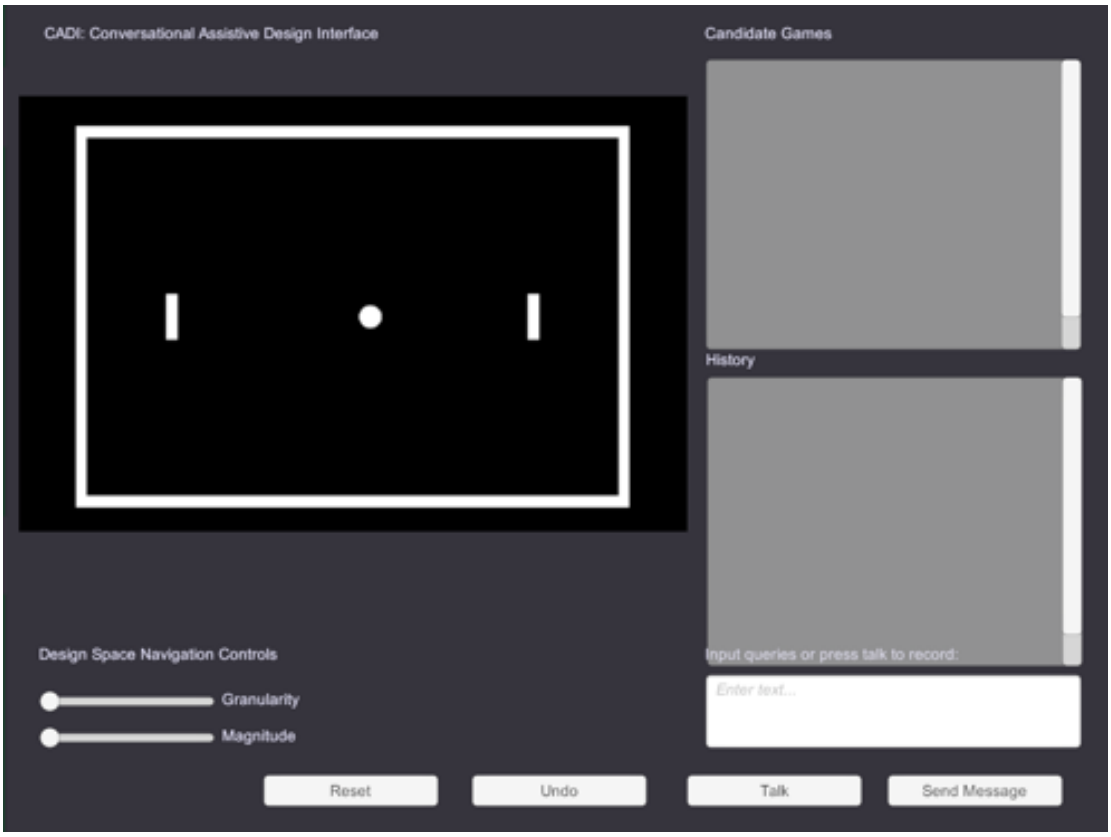


Figure 3.3: A screenshot showing the main UI for CADI

the design space of pong at a sub textual level. In addition, we utilize the concept of fluidic games [37] as our inspiration to translate the design space of Pong variations into a parameter-vector space over which we can map affective qualifiers from natural language into a series of compound features over which to explore the design space in. We followed Colton et al.'s [11] methodology in the design process of building our parameter space for Pong variations, and the resultant emergent-like compound features that arise from the concept.

With this justification, we built our parameter space in terms of core features of the games such as the number of agents (balls, paddles) in each variation, different

arena shapes, spatial features (position, rotation) for each agent. Furthermore, parameters were built from the affordances given by the Unity3D [57] engine such as physics parameters for rigidbodies and features specific to trail and particle systems embedded in the game. Finally, aesthetic features related to the game were defined in our implementation for CADI in terms of color palettes, particle systems for collisions, and trail renderers for the agents in the game. This resulted in 29 core parameters that were defined for the fluidic game design space of Pong variations.

After designing our core parameters for our design space, we proceeded to build our compound features that map to natural language qualitative terms for our system in order to address our research question of how to reason over designer intent in mixed-initiative PCG systems. For this purpose, we opted to author a mapping of grouped natural language concepts to compound features in the game. Our first step was to group words into a series of concepts that can be mapped to a feature in the game. For example, the concept for speed captures words such as fast, slow, quick, lethargic, amongst others. These concepts let us cover a wide number of possible scenarios in which the user wants to modify a feature within our design space. Table 3.1 shows a sample of word to compound feature mappings in CADI.

Compound Feature	Word Concepts	Core Feature Combinations
Vibrant	colorful, vibrant	Color palette Particle FX
Frantic	frantic, frenzied, hyper	Number of balls Ball speed Paddle speed Particle trails
Unfair	asymmetric, unfair, unjust	Paddle size Goal zone size (for only one player)

Table 3.1: A sample mapping of words to compound features and their associated core parameters in CADI.

3.4.2 Natural Language Understanding

We then authored a series of patterns in Chatscript that map to natural language queries about the game. These patterns can catch queries related to compound features such as "make the game more aggressive" or "make the ball less lethargic". In addition, we added patterns that let the user directly control the core parameters of our design space. An example of a direct control query would be "make the ball red". Our decision to add direct control queries stems from a necessity to cover a wide range of interactions that would be expected from a traditional GUI based mixed-initiative system. By allowing both control over compound features with qualitative terms, and direct control of core parameters in our design space, the user can explore the design space in a manner that reflects their intent to the best of their abilities.

Having designed our NLU component in Chatscript, we were informed by Colton et al.'s [11] methodology for creating the emergent-like compound features in our game. We built the compound features as a combination of features over

which CADI moves in the design space of Pong variations by building a series of example games that encapsulate the meaning of an compound feature. For example, when creating the compound feature for vibrancy we explored the design space by creating example games that capture the intent of the compound feature. After creating the game, we compared our example to an initial solution (original Pong) and determined which parameters changed in the game to capture the concept of vibrancy. In this case, the features modified were the color palette for the game, an application of extra saturation in the HSV color space for the palette, and the application of particle effects to the agents. This process resulted in the creation of 20 different compound features that map to the grouped concepts in our NLU module.

3.4.3 Moving in the Design Space of Pong Variations

After defining our parametric design space, a natural language representation, and a series of compound features that map to natural language concepts, we addressed the research question of how to translate natural language queries into a quantitative movement within the design space of CADI. This is important to address since natural language queries such as "make the ball go faster" or "make the game more colorful" can be ambiguous and the definition of faster or more colorful can vary between users. With this in mind we opted for a simplistic approach that covers a wide range of possibilities of moving along the design space

of our system. This was achieved by moving in "steps" along the parameter values for each compound feature. These steps are calculated by the following formulas:

$$step = m * (q * \frac{|max(param) - min(param)|}{granularity})$$

$$q = \begin{cases} 1 \rightarrow quantifier = "positive" \\ -1 \rightarrow quantifier = "negative" \end{cases}$$

$$m = \begin{cases} 1 \rightarrow unactivated \\ m \in \mathbb{Z} : m \in [2, 10] \rightarrow activated \end{cases}$$

The steps are calculated by taking the maximum and minimum for each parameter that is modified, and then divided over a user set granularity parameter that controls how much does CADI move within the design space of our features both core and compound. The magnitude and direction of our steps are calculated by the values q (for quantifier) and m (for magnifier). The quantifier is extracted from the natural language query in our NLU component with words like more meaning a positive movement in the design space, and less meaning a negative movement. The magnifier is a combination of a user set parameter and extraction from a natural language query. Phrases such as "even more" and "way less" are processed by our NLU component as a trigger for activating the magnifier variable in our step definition. The values for m and granularity are available to the user

within CADI's UI as a couple of sliders. We believe that by letting the user control the amount CADI moves within the design space affords a fine-grained control of the design space navigation that can suit the user's needs to the best of their ability. While this scheme is not ideal in the sense that it provides a fully autonomous way of understanding of how to map user queries to a magnitude and direction in the design space, we consider it a first step that might let us understand what users expect when providing natural language queries as movements in the design space of our system. Figure 3.4 shows some example games generated by CADI.

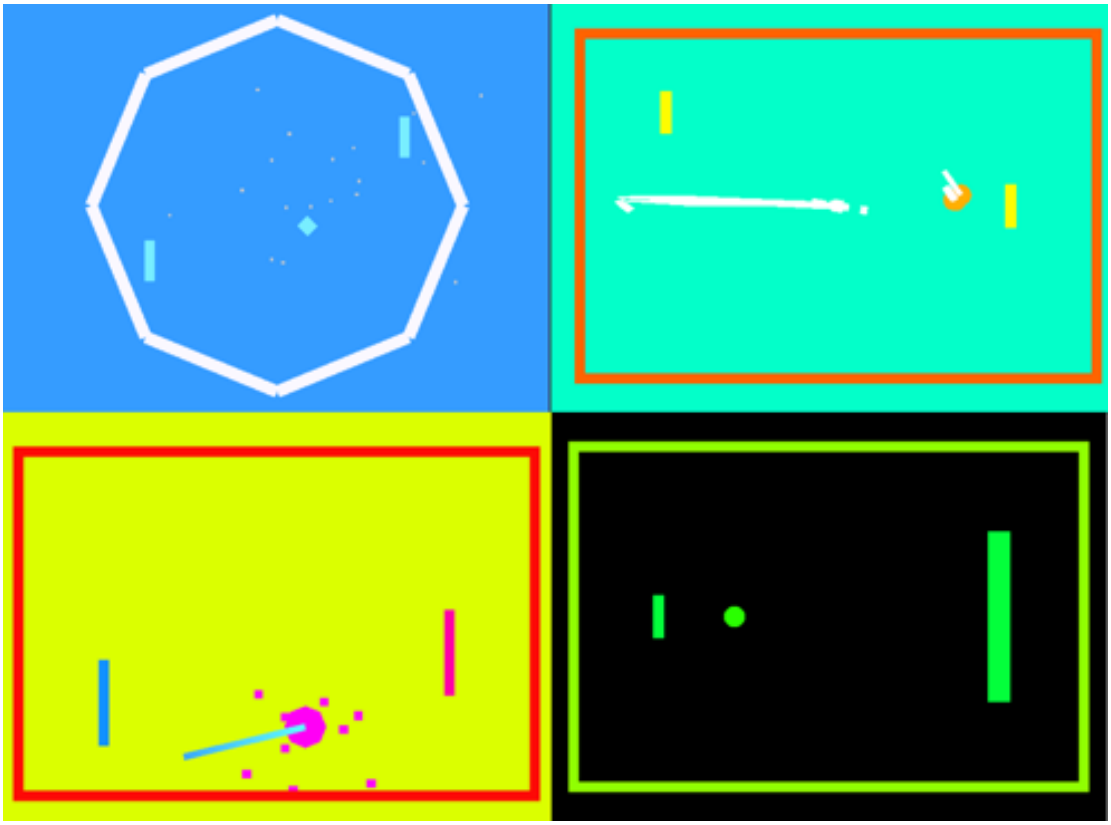


Figure 3.4: An example of generated output from CADI

3.4.4 Sample Interaction Session

Finally, in figure 3.5 we provide an example of how the user interacts with CADI and the internal steps on how each solution is generated in terms of natural language queries, and feature transformations.

State	Natural Language Query	Emergent Feature	Feature Transformation	Game
Initial	none	none	none	
Solution 1	"make the arena rounder"	round	agent: arena feature: n_sides(arena) quantifier: positive	
Solution 2	"make the game look cold"	cold	agent: game feature: color palette (from API) + particle effects quantifier: positive	
Solution 3	"make the ball less round"	round	agent: ball feature: n_sides(ball) quantifier: negative	
Solution 4	"make the game more frantic"	frantic	agent: game feature: n_balls + speed(ball) + speed(paddle) +trail_fx(ball) quantifier: positive	

Figure 3.5: An example of a sample creation session using CADI.

3.5 Conclusions and Future Work

We introduced CADI a natural language interface based mixed-initiative PCG system that is capable of exploring the fluidic game design space of Pong variations. We address a series of research questions arising from the implementation of a natural language interface within a mixed-initiative PCG system. These questions pertain to issues like understanding user intent in both quantitative

and qualitative terms, as well as providing efficient interaction modalities in our system. We discuss the methodologies and design decisions taken to address our research questions, as well as the details of the implementation of CADI in terms of its design space, knowledge representation, and architecture.

We believe CADI is a first step towards an exploration of different user modalities in mixed-initiative PCG systems. While this implementation addresses some initial questions on how to design a conversational system, and about understanding a design space in qualitative terms through the use of compound features we believe that there is a potential for exploring these issues further. Further work in this project includes an evaluation of CADI compared to a traditional GUI based mixed-initiative system, and an analysis of whether utilizing natural language as a driver for design space exploration addresses the research questions posed in this chapter.

Future directions in this field could involve addressing explainability and transparency in conversational driven mixed-initiative systems informed by Zhu et al.'s work [64] about explainable AI for designers (XAID). In addition, new techniques based on data-driven approaches such as utilizing and transforming open data from the internet [4] or the application of neural style-transfer techniques to in-game assets [10] could add new depths to this work.

Chapter 4

A Methodology For Developing Natural Language Interfaces for Procedural Content Generation

4.1 Introduction

In order to leverage the expressive power of natural language input in a PCG system, we first need to develop an understanding of the interaction modalities that natural language affords, the generative space over which the system acts, and what role the natural language interface takes in the co-creation process. By using a methodological approach, we can develop a natural language interface that captures a model of designer intent that uses descriptive language in terms of the

generative artifact such that the user is only exposed to the relevant aspects of the parameter space of the system rather than exposing all controls at once as used in traditional GUI based systems.

This methodology is composed of several steps that the researcher or technical designer needs to consider during the design phase of a PCG system in order to provide natural language input to their system. The steps in this methodology cover the following questions a designer needs to ask in order to create a mental model of designer intent that produces effective interactions with the system rather than replicating the same interaction modality of traditional GUI systems.

1. **Understanding the Interaction Model:** How do we interact with a natural language interface and how should the system react to commands?

2. **Understanding the Generative Space:** What are the sub-components of the generative artifact and what parameters map to which sub-component?

3. **Generating a Design Vocabulary:** What natural language descriptions can fit to which combination of parameters? Are there any exemplary points in the design space that fit a specific natural language concept?

4. **Design Vocabulary Expansion:** Are there other ways to communicate the same concept? How do we deal with counter-exemplars?

5. **Natural Language Query Development:** What actions do we allow the user to make? How do we deal with notions of magnitude?

We believe that by taking a methodological approach to designing a natural language interface for PCG systems, researchers and technical designers can incorporate new input modalities that ease the burden of searching a complex and high dimensional design space independent of what is being generated. This provides a framework upon which new and even existing systems can be augmented by the creation of a model of intent about the generator based on natural language descriptions of the output of the generator.

In this chapter, we provide a detailed exploration of how a technical designer can use this methodology to implement a natural language interface for mixed-initiative PCG systems. Such systems leverage the advantages of natural language interfaces to reduce interface complexity and model designer intent. To this end, we explore the following research questions:

- How do we define a design vocabulary for a generative system?
- How do we map concepts from a design vocabulary to a combination of controls such that the user can efficiently explore the design space?
- What conversational design patterns do we support in designing a natural language interface for PCG systems.

The following sections address these questions by providing a methodology that describes how technical designers and researchers can create meaningful natural language interactions for PCG systems destined for a beginner non-technical user.

In addition, we provide a case study of how this methodology can be implemented to modify shader-based materials in Unity3D with examples based on the Water4 shader. Finally, a discussion is provided of how the issues described above are addressed to create a natural language-based interface PCG system.

4.2 Related Work

The work described in this chapter is informed by developments in natural language interfaces for design, existing mixed-initiative approaches to PCG, and theory concerning the design of generative design tools. Several mixed-initiative PCG systems have addressed the problem of easing design space exploration. The typical approach provides a graphical user-interface which permits interaction and collaboration with the algorithmic back-end of the system, thereby supporting co-creation of a novel artifact. Notable systems in this area include Ropossum [50], Sentient Sketchbook [27], and Tanagra [53], which permit users to create levels for the game Cut the Rope [63], RTS games, and platformers respectively. Other developments in mixed-initiative PCG systems can be seen at the intersection of Automated Game Design (AGD) and mixed-initiative PCG. Some exemplary systems in this field are Cillr [36, 37] and Wevva [45]. In Cillr and Wevva, users are provided with the tools to create small games on a mobile device.

Natural language inputs to design tools have been explored in the area of AI-assisted design. The AGD system ANGELINA-3 uses natural language input

from the user in which meaning is extracted to create a series of aesthetic elements for generated games. Words are mapped to related concepts mined from the web, thereby creating necessary assets such as textures, music, and commentary for a generated Metroidvania style game. Other AGD systems that use a natural language input are Interactive Game-Design Assistant [40] and Game-o-Matic [60] which take a natural language representation of a series of concepts and relations, that are in turn are mapped to a series of game mechanics to generate Warioware Inc. [55] style microgames. Outside of game design and PCG, there are design systems that operate over natural language input in order to generate different types of artifacts, as well as constraints. For example, Cheong et al. [9, 23] convert natural language input into design constraints in mechanical CAD applications. Firedrop.ai's system Sacha [16] uses natural language input to generate websites, guiding the designer using a conversational interface.

Our work is informed by the concept of Casual Creators by Compton and Mateas [12], specifically their casual creator design patterns. We draw inspiration from design patterns such as "providing instant feedback", "avoiding a blank canvas initial solution", and "modifying meaningful parameters" within the design space. By applying these design principles to natural language interfaces in PCG, we can provide an experience that allows non-technical users to engage with generators that can be perceived as technically daunting.

Furthermore, we are informed by Colton et al.'s [11] parameter design method-

ology for casual creators. Their methodology provides a set of best practices on how to design a parameter space in the context of PCG tools, such as creating exemplars that capture special points of interest in the design space, the separation of parameters according to functionality, and the creation of "emergent features" by combining parameters that allow multi-dimensional design space movements. In addition, this methodology provides grounding on how to understand the design space of a generator from a parameter design perspective, compared to approaches that focus on the analysis of a generator's output.

4.3 Methodology

In order to understand how to implement a natural language interface for a PCG system, we must first address the interaction model inherent to such an interface. Compared to the traditional "freeform" nature of a graphical user interface containing a different control for each parameter in the design space, a natural language interface affords a more linear mode of interaction analogous to a conversation between the user and the procedural generator. In addition, whereas in a GUI the user would be presented with a number of graphical controls (sliders, text boxes, toggles) that control each parameter in the generator, a natural language interface might not expose the user to any controls at all. As such, the system is given a larger degree of autonomy to determine how the designer's actions are interpreted as compared with direct manipulation of the

controls. With these considerations in mind the designer has to understand how the interface controls the generator to design meaningful interactions that provide the user with a simplification of their workflows rather than an simple translation of traditional control manipulation via natural language commands.

4.3.1 Interaction Model

The interaction model that the user (e.g., a technical artist, or a gameplay programmer) of a natural language interface for a PCG system would use can be distilled into the five following steps (see Figure 4.1).

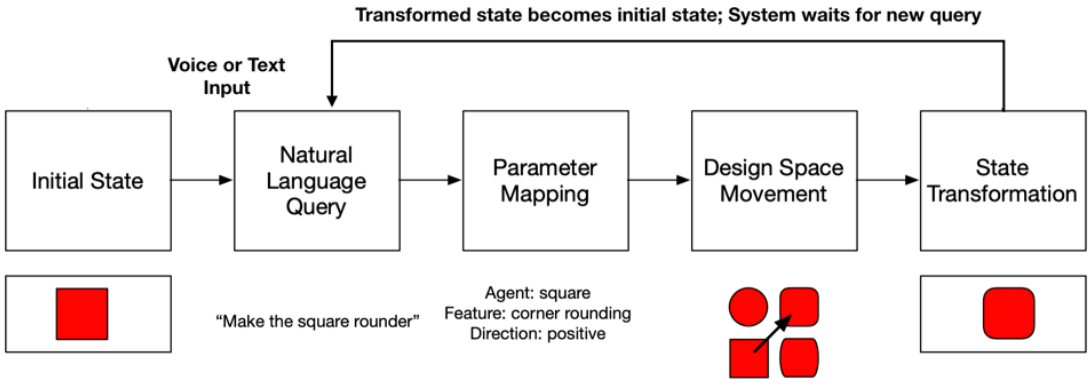


Figure 4.1: Architecture diagram for a natural-language interface for PCG Systems

1. **Initial State:** An initial point of reference is presented to the user to avoid the problem of starting with a blank canvas. Compton and Mateas [12] mention the importance of providing an initial point in the design space for users to work with as a way to reduce the "terror that comes from facing

a blank canvas". This is an important consideration since, compared to traditional GUIs where there is a visible control to modify the artifact right, natural language UIs usually await a first command from the user. Showing an initial example of the generative space can alleviate some of the difficulties the user might experience when interacting with the system for the first time. Depending on the artifact to be generated, the designer of the system can choose an inspiring example or allow the user to choose an example from a series of reference points (e.g. sample output chosen by the tool's developer) in the design space.

2. **Natural Language Query:** The user inputs a natural query related to the artifact. This can be done via voice or text. These queries are the commands that the user makes to the system analogous to modifying a control in a GUI. For instance, a query can be in the form of: "make the artifact/sub-component quantifier concept". The structure of accepted queries can be authored by the system's designer or used as unstructured text. This query is passed to a Natural Language Understanding (NLU) unit for interpretation.

3. **Parameter Mapping:** The NLU component interprets the natural language query and then maps it to a combination of features in the design space. For example, in the implementation of CADI [30], a concept such as "cold" maps to aesthetic features in the design space such as the color palette (bluer tones) and particle effects (trail rendering systems, particles

on collision), whereas a concept such as "unfair" affects mechanical features in the design space such as the size of the paddles (scale of paddle in the game world) in the game. Additionally, the NLU component can understand the direction in the design space by means of a quantifier keyword such as "more" for a positive direction, and "less" for a negative direction.

4. **Moving in the Design Space:** Having extracted the affected features and the direction of movement in the design space, the system applies the transformation of the current solution and retrieves the new point in the design space. The magnitude of the jump in the design space can be either left to the user's desire by providing controls for the sensitivity of movement in the design space (e.g. controls for how much does a parameter change) or using assumptions made by the system designer (e.g. using a constant rate of change for certain parameter, setting upper and lower values to parameters). In addition, the UI could provide exposure to the relevant parameters in the form of GUI controls to provide further fine-tuning capabilities to the user.

5. **New State Loop:** The transformed solution is presented to the user, and it becomes the current state. Flow now loops back to step number 2, as the system uses awaits a new natural language query.

Having described the interaction model for a natural language interface in PCG, we now detail the methodology that the designer of a PCG system would

use for creating the NLU component, feature mapping, and design space movements. Figure 4.2 provides an overview of the methodology steps described in the subsections below.

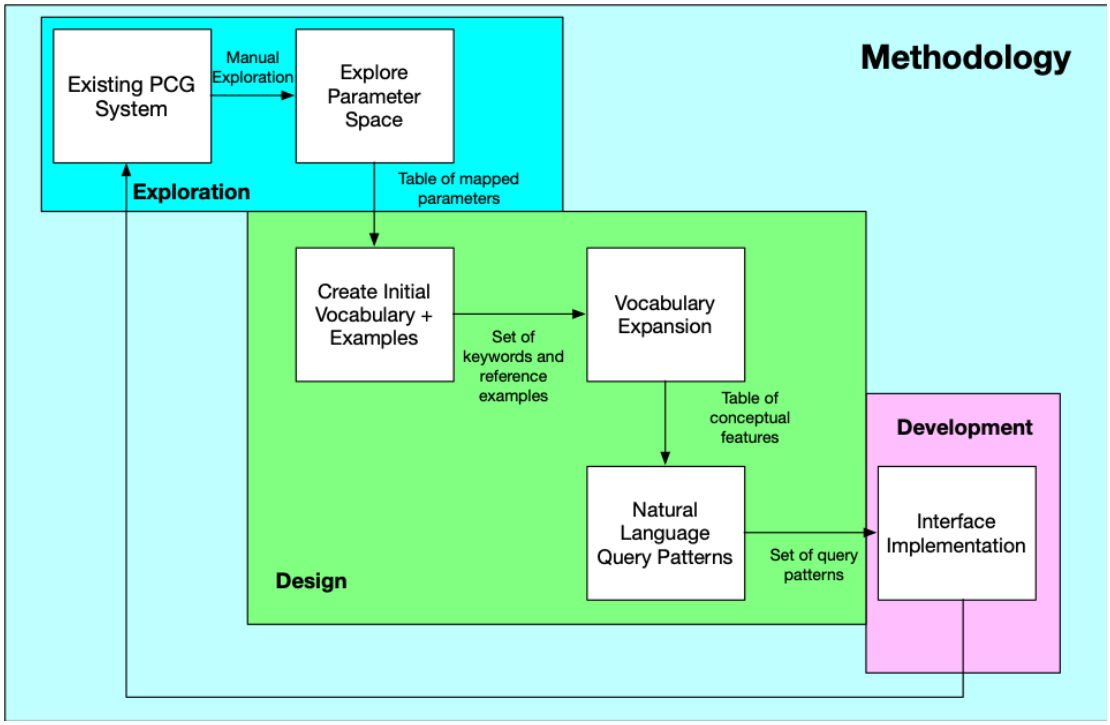


Figure 4.2: Process diagram for the methodology used to develop a natural language interface for PCG systems.

4.3.2 Exploring the Parameter Space and Creating an Initial Vocabulary

The first and most important step in the creation of a natural language interface for a PCG system is to understand the nature of the generated artifact by manually exploring the parameters of the procedurally generated design space, after which a series of natural language keywords can be applied to a combination

of features in the design space.

When exploring the parameters of the design space, it is important to understand what is being generated in terms of the sub-components of the generated artifact. The separation of the generator into its subcomponents. This lets the designer understand what characteristics of the artifact can be modified by which parameter (or combination of parameters) and what type of modification in the design space is needed. A first step would be to understand the nature of the output, whether it comprises only one type of output (sound, visual) or more complex output such as a small game with multiple agents. Separating concerns in this step by sub-component allows to understand the range of actions that can be taken in the tool such as changing colors, or modifying a procedural animation which require different types of modifications in the design space as well as different vocabularies to perform this kind of action.

An example can be seen in an implemented system such as CADI [30]. In this case the subcomponents are the balls, paddles, and the arena for each Pong variation. After separating the generator into its functional sub-components, we now can explore and map the features that comprise them, such as aesthetics, and mechanics. Features that control colors or particle effects can be mapped as aesthetic characteristics, whereas physics parameters can be assigned to mechanical characteristics. This understanding of what aspect of the generator and what sub-component maps to what functionality of the generator can provide an initial

ontology for the design vocabulary that can be generated.

Having mapped out the functional aspects of the generator into sub-components, a series of reference points based on natural language keywords can be made. These reference points can be seen as an initial mapping of our keywords in the design vocabulary to a specific point in the design space. This will comprise our initial design vocabulary. This is similar to the "capture an inspiring example" phase in Colton et al.'s [11] methodology. The first step in this phase is to find the desired keywords associated with the generated artifact. In the case of CADI [30], the basis for the design vocabulary was Swink's Game Feel theory [19] which focuses on the affective experience elicited by a game's design. This resulted in a design vocabulary, based on affective keywords, that could map to the generated artifact's control scheme, physics, and aesthetics. Some keywords associated with the initial design vocabulary for CADI were "cold", "unfair", and "sticky". In systems that deal with one specific type of artifact (such as asset generators) the initial design vocabulary can be defined using adjectives commonly applied to the represented artifact. One example would be to describe a tree with words such as "leafy", "twisted" or "blossoming", where as an ocean water generator could use words like "calm"," turbulent" or "tropical". In this phase, the end-users can be queried about how they would describe the generated artifacts by showing them random or manually selected reference points in the design space and asked to describe some of the qualitative characteristics of the artifact. This can be done

via a user study with potential end-users of the system, or via crowdsourcing platforms.

With this initial set of descriptive keywords, the PCG system designer can now create a reference example in the design space that captures the meaning of the keyword. A reference example is a specific point in the design space that represents an archetypical artifact described by the keyword. While creating these examples it is important to record what variables influenced the design space exploration the most.

4.3.3 Expanding the Vocabulary and Building Conceptual Features

Now that an initial design vocabulary has been set for the generated artifacts, we can expand the vocabulary to provide more options for expressing desired intent about the artifact. One of the simplest ways to expand the vocabulary is synonyms of the initial keywords in our initial design vocabulary. Synonyms allow the system's designer to enhance the NLU component's capabilities to detect intent for similar concepts, as the mental model of what language to use to describe an artifact may vary from user to user. By grouping similar words to one single concept, the designer can cover many similar use cases.

Another technique is to create a counter-example for each initial keyword. This approach uses the same features, but with values mapped to the antonym of the

original keyword. This technique allows for the creation of a continuous design axis. A benefit of this is it allows the user to explore a large combination space of desired feature combinations without having to tweak one parameter at a time. In addition, synonyms can be added to the counter-example as an expansion of the concept. One such example can be seen in CADI [30], where the words "warm" and "cool" form a single natural language concept that modifies the color palette between blue and red hues. In this case, the "cool" example was defined in the initial exploration of the design space, upon which a counter-example ("warm") was worked using the same set of modified features. In addition, synonyms were used (e.g "cold", "frigid", "icy", "hot", "fiery", "blazing") to expand the design vocabulary in order to allow more options for the user to express their intent.

Tying both the design vocabulary generation and expansion we arrive at a series of mappings of natural language keywords to a set of compound core features for our generator. We refer to these mappings as conceptual features in our natural language interface design. These conceptual features conform the core concept for using natural language in PCG systems as they capture different intents from the system's users.

4.3.4 Developing Natural Language Query Patterns

Having defined a set of conceptual features representing the combinations of features that describe an artifact using expressive language, the next step is the

development of patterns for user queries through the natural language interface. These queries are what allow users to perform actions on the design using the conceptual features that were defined in the previous phase. One initial strategy is to form a series of patterns in the form of a declaration about the artifact and its identified subcomponents using the conceptual features. These patterns take the following form:

1. A noun referring to the artifact or a sub-component: In CADI [30] it could be the entire game, the paddles, or the Pong ball.

2. A word that signifies the direction of movement along the axis of a conceptual feature: Words like "more" or "less" are good starting points to signify a positive or negative movement in the design space. We call these words quantifiers and address their handling later in this subsection.

3. Finally, a conceptual feature to modify in the design space.

Following this rule, we end up with queries such as "make the artifact/subcomponent more/less conceptual feature" which provides the user with a set of initial exploration actions. While this pattern allows for a "quick and dirty" approach to moving in the design space it does not allow for more fine-grained control of design space exploration. This is because this pattern relies on default assumptions set by the system's designer about how much or little the quantifier moves in the design space.

A different approach to learning what kind of natural language queries might arise when interacting with the PCG system is to have potential end users of the system to participate in a "Wizard of Oz" type user study. In this study, the users are asked by the researcher or technical designer to interact with an "idealized" version of the natural language-based interface. This can be either the researcher manually manipulating the PCG system behind the scenes, or by a paper prototype in which the researcher or designer shows the changes to the artifact manually on the prototype. This type of approach while time consuming, might enable the researcher or technical designer understand what kinds of emergent queries outside of their mental knowledge might occur when interacting with the tool. In addition, this approach might also help enrich the vocabulary as new words outside of what has been authored up to this point can be added based on user feedback.

We would suggest that a hybrid approach to this task might yield the best results as the queries designed by the researcher or technical designer get to be tested in the user study, while learning new emergent queries that potential users of the system might use when interacting with our natural language-based interface.

Now that we have an initial set of actions to operate in the design space using expressive language, we can expand our patterns to handle more specific cases. This is done by using verbs to map the natural language query to specific functionalities in the generator. For example, the phrases “look like” affect visual

parameters in the design space, while phrases like "move like" affect motion components in our generator. In addition, we can constrict the set of conceptual features that map to each verb, given that a mapping of conceptual features to functionalities has been explored in previous phases of the design of the natural language interface.

Having established some initial patterns defining general actions and functionality-specific actions, we address the need for two specific use cases related to more direct control of the generator. One is numerical control of our conceptual features, and the other is direct manipulation of the parameter space in numerical terms. Our first pattern is to handle cases in which a user can say "Make the artifact/sub-component X percent more/less conceptual feature". In this case, we want to keep track of where in the design space the user is located to apply a transformation over the values affected in the conceptual feature understood from the query. Our second pattern is to allow direct control of the feature space to the user. This can be done by simply mapping named keywords for each feature in the design space to allow a straightforward manipulation of the parameter space for more advanced users that need fine-grained control to fine tune their expected artifacts. However, this runs counter to our goal of having conceptual terms which map across multiple controls at the same time.

Finally, we address the concept of quantifiers, which are keywords that represent the direction of movement in the design space for a conceptual keyword. We

mentioned the usage of the keywords “more” and “less” as signifiers of positive and negative movement in the design space of a conceptual feature. To control this movement, we develop an approach for translating keywords into a numerical “step” movement in the design space, as described in the following equation from the previous chapter.

$$step = m * (q * \frac{|max(param) - min(param)|}{granularity})$$

$$q = \begin{cases} 1 \rightarrow quantifier = \text{"}positive\text{"} \\ -1 \rightarrow quantifier = \text{"}negative\text{"} \end{cases}$$

$$m = \begin{cases} 1 \rightarrow unactivated \\ m \in \mathbb{Z} : m \in [2, 10] \rightarrow activated \end{cases}$$

In this equation, the granularity parameter controls the amount of movement desired in each “step”. The quantifier q (positive or negative) indicates the desired direction of movement. A magnifier parameter m accounts for more dramatic jumps in the design space, which is mapped to phrases such as “even more” or “way less”.

The user is typically presented with the option of setting the amount of granularity and the value of the magnifier parameter via two graphical user interface controls. This allows the user to determine and fine-tune the amount of change to

best reflect their idea of the magnitude of movement in the design space. Alternatively, the designer can make assumptions of how much movement is expected from the model and further reduce interface complexity for the user.

4.3.5 Addressing Design Concerns

In summary, our methodology for creating a natural language representation of a PCG system design space begins from an initial exploration of the design space, proceeds to the development of a design vocabulary and conceptual features, and ends with development of the patterns for the natural language queries that are used to represent actions in the PCG system. We now address the emergent design issues that arise from such development, including words outside the design vocabulary and avoiding "junk" output. While our design vocabulary and conceptual features might be able to capture the users' intent to a certain degree, expressive language queries can contain concepts that are outside the set of conceptual features. To avoid problems with a lack of feedback due to unrecognized keywords, a fallback strategy is needed. One such strategy is to use web data to define a default set of features. One example of this style of fallback strategy was implemented in CADI [30], where unrecognized color keywords led to a call to a RESTful API from colourlovers.com [33], which returned a color matching the keyword. By using this strategy, designers can have a larger set of keyword coverage beyond their defined design vocabulary and conceptual feature set with-

out sacrificing feedback for users when interacting with the system. Another issue is finding regions of "junk" output. This can be defined as regions in the design space of the generated artifact that are not consistent, present erratic behaviors, or are considered to be aesthetically unpleasant. This can be addressed during the explorative phase of the methodology by capturing what are considered "acceptable" values by setting upper and lower bound values for each explored parameter. While this strategy might reduce the number of solutions in the design space that can be explored, avoiding "junk" output can help achieve a sense of consistency in our natural language-based interface system.

4.4 Use Case Implementation: Modifying the Water4 plugin for Unity3D

Having described our methodology for creating a natural language-based interface for PCG, we now show an example implementation made for an existing plug-in in the Unity3D game engine called Water4[56]. Water4 is a shader-based plugin for the Unity3D engine that creates volumes of water and uses a Gerstner-based wave simulation to animate the water mesh it generates.

4.4.1 Design Space Exploration and Initial Design Vocabulary

We followed the methodology presented above to generate the conceptual features and natural language-based queries for interaction with Water4. The first step was to understand the generative design space of the plug-in and to separate the sub-components and parameters based on their functionality. This was done by a manually exploring the design space of the plug-in, modifying one parameter at a time. In this case we identified the following sub-components in the generated artifact: a water mesh, a shader for the water, a shader for foam, and a Gerstner displacement-based simulation for the waves. Upon investigation of the plug-in's parameters we identified 46 different modifiable parameters: 18 for modifying the water mesh and foam, and 24 for modifying the wave simulation. Table 4.1 shows the mapping between parameters and functionality in Water4.

Now that our parameters space has been mapped out by functionality and artifact sub-components, we establish our initial design vocabulary. This was accomplished by manually creating examples that represented different aspects of the artifact, such as different color palettes and wave behaviors. Sets of keywords that described the ocean, waves, and water were also recorded. Some examples of the initial design vocabulary were words describing the ocean such as "turbulent", "calm", and "Caribbean".

This led to the creation of 10 initial examples mapped to a keyword, with

Component	Parameters	Functionality
Water Mesh	Tiling1/2 (Vector2 x2) Direction 1/2 (Vector2 x2)	Visual
Water Shader	Refraction Color Reflection Color Texture Normals (float x3) Fresnel (float x3) Fading (float x3)	Visual
Foam Shader	Foam Texture Foam Intensity (float) Foam Cutoff (float)	Visual
Wave Simulation	Gerstner Amplitude (Vector4) Gerstner Steepness(Vector4) Gerstner Frequency (Vector4) Direction Scale 1/2 (Vector4 x2)	Motion

Table 4.1: Mapping the parameter space in Water4 by functionality and sub-component.

associated parameters recorded. In this case the word "Caribbean" maps mostly to the color of the water rather than the shape of the waves, whereas examples like "calm" and "turbulent" are mostly related to the behavior of the wave simulation. Figure 4.3 shows some exemplary artifacts in Water4 for the keywords used in this paragraph.

4.4.2 Design Vocabulary Expansion

Starting from these examples, we expanded the design vocabulary by using synonyms to create additional positive instances of the conceptual feature. For example, when designing the "calm" example we noticed that the wave simulation parameters comprised the set of modified features for our generated artifact's wave

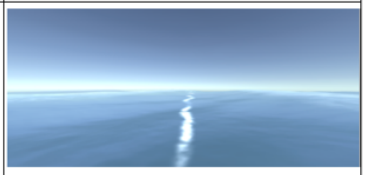
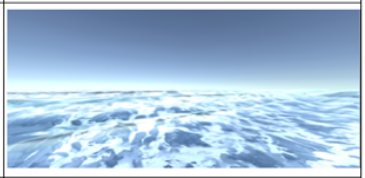
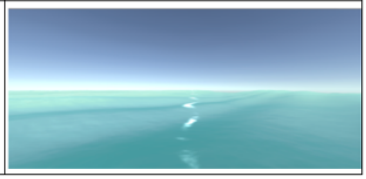
Keyword Exemplar	Used Parameters	Output
Calm	**Gerstner Speed Low** (values = 1,1,1,1) **Gerstner Amplitude Low** (values = 0.2, 0.2,0.2,0.2)	
Turbulent	**Foam Intensity High** (value = 0.869) **Gerstner Amplitude Med-High** (values = 0.5,0.7,0.3,1) **Gerstner Speed High** (values = 1,3,4,3) **Gerstner Steepness X high** (value = 3)	
Caribbean	**Foam Intensity Low** (value = 0) **Water Shader Colors:** (Green-blue hues)	

Figure 4.3: Initial design space examples in Water4 with associated parameter values.

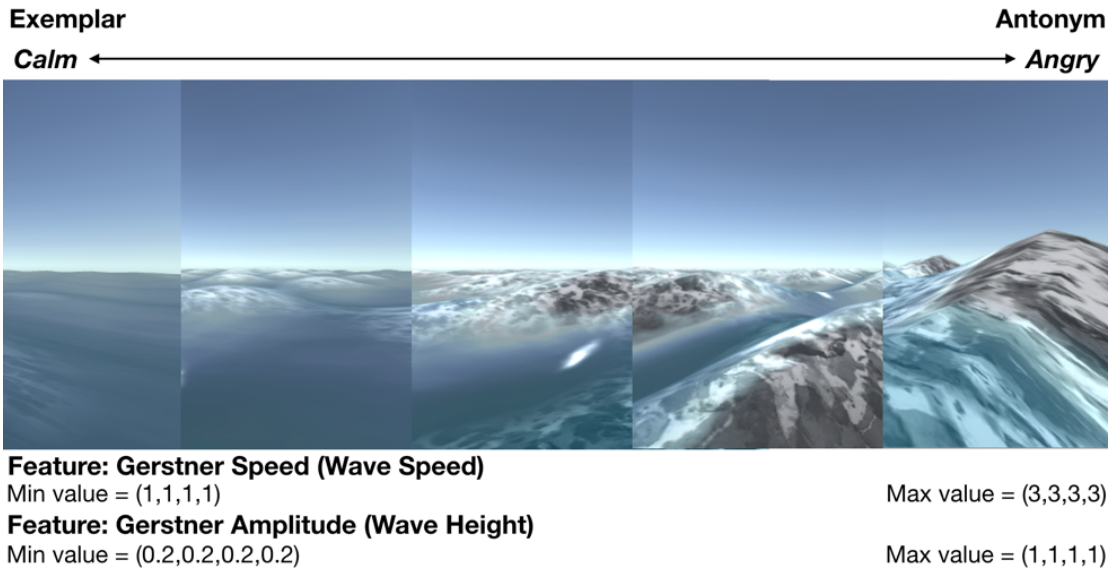


Figure 4.4: The resultant design axis for "calm" and "angry" in Water4.

simulation subcomponent. With this in mind, we generated a counterexample for "angry" using the same features in order to create a continuous design axis that

allows the user to move in the design space between “angry” and “calm” while using the same set of generator features. Figure 4.4 shows an illustration of the design axis generated for “calm” and “angry” in Water4.

Following this example, we proceeded to develop each keyword of our initial design vocabulary by using synonyms and antonyms to create our conceptual feature set. Table 4.2 shows a sample of our expanded conceptual features for the examples in Figure 4.3.

Conceptual Feature Name	**Word Concepts (Positive and Negative)**	**Parameter Combinations**
Calm-Angry	Calm, tranquil, peaceful Angry,tempestuous,wild	Gerstner Speed Gerstner Amplitude
Clear-Turbulent	Clear,clean,pristine Turbulent,murky,foamy	Foam Intensity Foam Cutoff
WaterTemperature	Caribbean,tropical,warm Atlantic,frigid,cold	Water Shader Colors (green and blue channels)

Table 4.2: Sample conceptual features used in our Water4 natural language interface.

4.4.3 Natural Language Query Patterns

Following the development of our conceptual feature set, we proceeded to build the NLU component in Chatscript [62]. The way Chatscript operates is by matching authored patterns for natural language queries. These patterns are a series of keywords that are defined in our NLU component that are arranged in a specific order. In this component we added the definitions for our design vocabulary grouped by conceptual features, the query patterns for controlling

conceptual features, and query patterns for direct control of the parameter space in Water4. Figure 4.5 shows an example of a query pattern and a definition for a concept using keywords in our design vocabulary.

```
concept: ~agents (water ocean)
concept: ~quantifier (more less)
concept: ~calmpos (calm peaceful tranquil)
#Sample Pattern

#Make the (agent) (more/less) (calm positive)
u:(* _~agents _~quantifier _~calmpos)
      $$agent = ^original(_0)
      $$quantifier = ^original(_1)
      $$qualifier = ^original(_2)

if($$quantifier == more){
^'{"feature": "calmpos", "quantifier": "positive", "agent":"$$agent"}'}
else{
^'{"feature": "calmpos", "quantifier": "negative", "agent":"$$agent"}'}
```

Figure 4.5: Sample Chatscript implementation of natural language query patterns.

4.4.4 Implementation Details

After developing our NLU component in Chatscript, we developed the following components in Unity3D following the architecture in Figure 4.6: A parameter modification script, an implementation of the quantifier model and a connection script to Lahar [16], a REST API that handles the Chatscript connection for our NLU component. Finally, we developed the graphical user interface that allows natural language-based interaction for Water4. In this step, we added controls for

voice recognition, textual input, controls for the manipulation of the quantifier model (granularity and magnification), and an option to save their final artifact such that it can be loaded into other Unity3D projects.

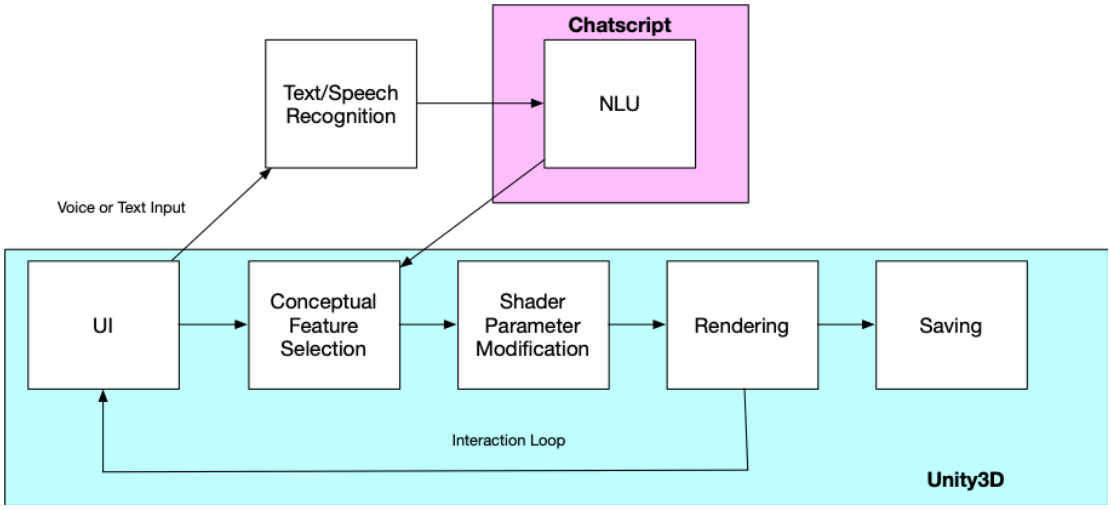


Figure 4.6: Architecture Diagram for our Water4 example implementation

With this, we ended up with the final implementation of our system as evidenced in Figure 4.7.

4.4.5 Addressing Design Issues

We now address the issues discussed in the introduction concerning blank canvas issues, interaction attrition, and linear workflow issues. To address our blank canvas issue, we set the initial state of the generator to a default value provided by the Water4 designers. This avoids the problem of starting at an extreme point in the design space (such as all parameters set to zero). By presenting the user with an internally consistent default configuration they are able make an initial

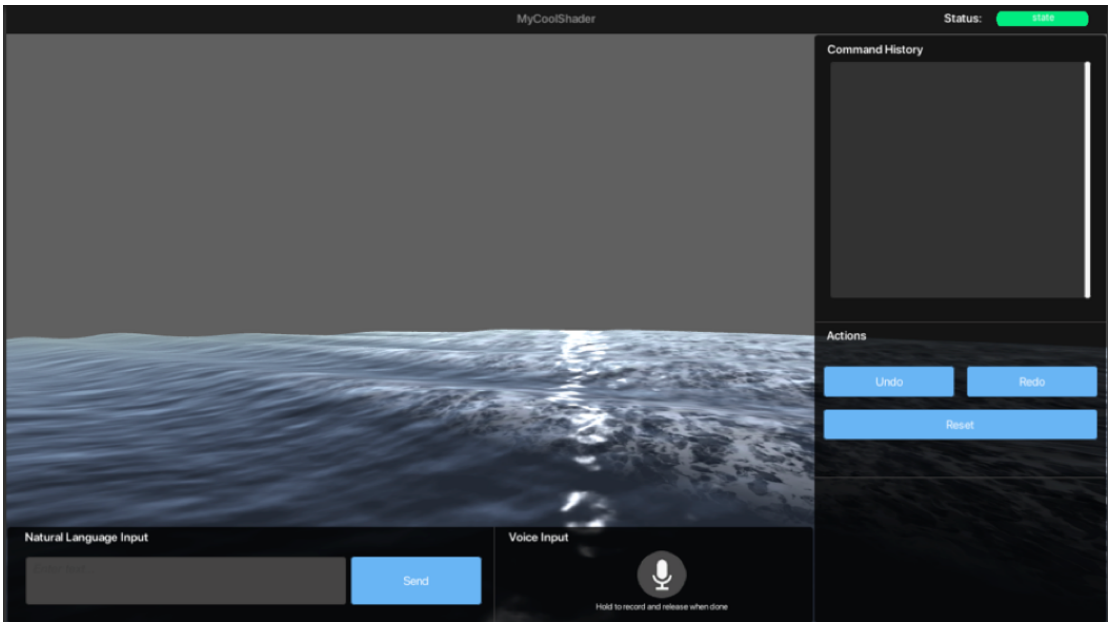


Figure 4.7: Screenshot of the implemented UI for our natural language interface for Water4.

query that takes them closer to their desired solution in the design space.

We address the interaction attrition issue by allowing the user to trace their steps back in the design space and by showing their history of interactions. In this way they can easily correct undesired moves in the design space without having to restart from an initial default state. We believe that keeping track of states and providing mechanisms to move within the interaction history allows the designer to efficiently explore the design space as they can explore different pathways without having to start over again every time the output is undesired.

We address linear workflow issues by allowing direct manipulation of the parameters in Water4 using natural language queries. This is seen as a fallback strategy analogous to direct modification of the plug-in. This type of interaction

allows for a more freeform style exploration in conjunction with our implementation of design space exploration history. Direct parameter manipulation allows the designer to freely move between previously visited points and use fine-grained controls to explore the design space without having to recur to expressive language as their only interaction option.

4.4.6 Example Interaction Session

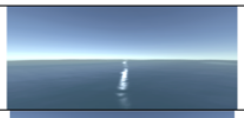
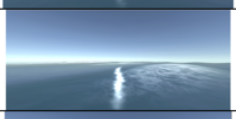
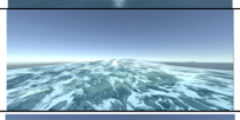
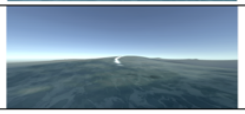
State	Natural Language Query	Conceptual Feature	Feature Transformation	Output
Initial	***none***	none	none	
Solution 1	***"make the water more stormy"***	Calm-angry	feature: GerstnerSpeed, GerstnerAplitude quantifier: positive	
Solution 2	***"make the waves taller"***	Height	feature: GerstnerSteepness quantifier: positive	
Solution 3	***"make the foam less turbulent"***	Clear-Turbulent	feature: FoamIntensity, FoamCutoff quantifier: negative	
Solution 4	***"make the water less warm"***	WaterTemperature	feature:ShaderColor quantifier: negative	

Figure 4.8: Example session showing natural language interaction with Water4.

Finally, we provide an example session of how a designer can interact with our user interface in accordance with the interaction model shown in Figure 4.1. In Figure 4.8 we show how a user can interact with our systems interface by using natural language queries, and the internal steps in our pipeline as the user moves through the design space of Water4, such as what conceptual features are chosen

in the NLU component, and the associated feature transformations and the output shown to the user.

4.5 Expanding Use Cases

The section above shows a concrete implemented example in the form of WATER4-NL a natural language interface for the Water4 plugin of the Unity3D game engine. In this section we showcase examples of how the natural language interaction paradigm explored in this paper can be implemented in different types of mixed-initiative generators with different levels of initiative taken by the procedural generator. We focus on three potential cases: an interactive evolution system similar to Picbreeder [48] that creates tracks for a racing game, a two-tiered texture generator that uses style transfer techniques and a parametric texture generator. Finally, the last use case is a MIDI-based audio track generator. It is worth noting that these examples are more of a "vision" of how natural language input can be used in the context of procedural content generation beyond the prototypes built for CADI [30] and WATER4-NL.

4.5.1 Interactive Evolution

Our first example is a track generator for an arbitrary racing game. In this case our system uses a different interaction paradigm to the one in section 4.3.1. Similar to Picbreeder [48] and the work in [7] our fictional track generator first

State	Natural Language Query	Conceptual Feature	Feature Transformation	System Response	Output
Initial	***none***	none	none	"Please choose a style of track to work with"	
Solution 1	***"I want to make a circuit"***	Track Type	feature: Track Type value: Circuit	"Here's some examples of a circuits, choose the one you like the most to begin"	
Solution 2	***"I like the first one but make it curvier"***	Solution Location + Curviness	Feature: Solution Location Value: 0 Feature: Curviness quantifier: positive	"Based on your command here's some potential solutions"	
Solution 3	***"I like the third one and the second one"***	Solution Location	feature: Solution Location value: [1,2]	"Making more tracks that look like your favorite options"	
Solution 4	***"I like the first one but make it faster"***	Solution Location + Density of Straight lines	Feature: Solution Location Value: 0 Feature: Straight Density quantifier: positive	"Here's some options that match your request"	
Solution 5	***"I like the first one"***	Solution Location	Feature: Solution Location Value: 0	"Choosing the first example"	

Figure 4.9: Example session showing natural language interaction with our hypothetical race track generator.

presents a set of potential solutions to the user to get a sense of what regions of the design space the generator should search in. After this step, the system presents a new set of solutions upon which the user can provide natural language descriptions of the ones they prefer. One could imagine the user presenting queries such as "Like the third one but curvier" or "Make it like the second track with a longer stretch". This way the generator takes the specific solutions ("the third one", "the second track") but also a set of constrains ("longer stretch","curvier") of what sections of the design space carry a larger weight in the evolution process in descriptive natural language terms. Once these constraints are accepted by the system the generator returns a set of solutions that best fit the constraints that the user provides, repeating the interaction loop until the user is satisfied

with the generator's output. Figure 4.9 showcases an interaction session of how an interactive evolution paradigm can be fit into a natural language interface implementation.

4.5.2 Two Different Levels of Initiative

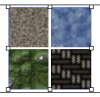
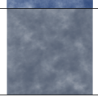
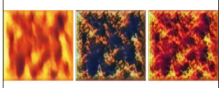
State	Natural Language Query	Conceptual Feature	Feature Transformation	System Response	Output
Initial	***none***	none	none	"Please choose a style of texture to work with"	
Solution 1	***"I want to make a skybox"***	Texture Type	feature: Generator type value: clouds	"Here's some examples of a sky, choose the one you like the most to begin"	
Solution 2	***"I like the first one"***	Solution Location	Feature: Solution Location Value: 0	"Choosing the first example"	
Solution 3	***"make the sky more gloomy"***	Sky color (mood)	feature: Background color saturation quantifier: negative	"Lowering background color saturation"	
Solution 4	***"make it look like it's on fire"***	Overall Style	feature: Style keyword, value: fire	"Here's some options that match your request"	
Solution 5	***"I like the second one"***	Solution Location	Feature: Solution Location Value: 0	"Choosing the second example"	

Figure 4.10: Example session showing natural language interaction with our hypothetical texture generator.

Our second example is a parametric texture generator which has a data-driven style transfer component to create visual variations of the output of the procedural texture generator. In this case, our interaction paradigm is purely reactive to user input for the texture generator, but the style transfer component the generator takes the initiative to find variations of the texture. For example, in this case the user can make queries about the texture generation process that result in a reaction

from the procedural generator similar to the WATER4-NL case. Queries such as "make the background color green" or "make the texture noisier" are mapped to parameters for the texture generator. But when the user chooses to stylize the texture, after generating their base sample, the system takes a larger amount of initiative. For instance, the user could tell the system "make it look like it's made out of wood" or "make the texture more like fire". For these queries, the style transfer component generates several options for the user to pick from. In this case, the options can be different samples based on different images mined from the web, or different samples generated from one image mined from the web. After choosing a sample, the user can either further refine the search by generating new samples based on the options the user prefers, or accept a solution, or do a new style transfer (using natural language input) over the original generated sample. This hypothetical system demonstrates a use case in which the same type of input can be used in a pipeline that integrates two different levels of generator initiative. An example interaction section for this system can be seen in figure 4.10. The textures were generated using Christian Petry's online texture generator.[44] The style transfer textures were generated using Reiichi Nakano's "Arbitrary Style Transfer in the Browser" [34].

State	Natural Language Query	Conceptual Feature	Feature Transformation	System Response	Output
Initial	***none***	none	none	"Please choose a style of song to start with"	
Solution 1	***"I want to make a techno song"***	Song Type	feature: Song Type value: Techno	"Here's some examples of a techno songs, choose the one you like the most to begin"	 3 audio tracks with a preview
Solution 2	***"I like the first one"***	Solution Location	Feature: Solution Location Value: 0	"Based on your command here's some potential solutions"	 Loaded main editor screen
Solution 3	***"Make the drum track louder"***	Track Volume	feature: Track Location value: Drums feature: Track volume Value: louder	"Adjusting drum track volume"	 Louder drum track
Solution 4	***"Add a piano track"***	Track addition	Feature: Track addition Value: 1 Feature: Track Instrument Value: Piano	"Here's some options that match your request"	 3 piano tracks with a preview
Solution 5	***"I like the third one"***	Solution Location	Feature: Solution Location Value: 0	"Choosing the third example"	 Editor screen with added piano track

Figure 4.11: Example session showing natural language interaction with our hypothetical texture generator.

4.5.3 Action-Dependent Initiative

Our final example is a music generator. In this case we envision a system in which the user can create a music track and edit its components (tempo, pitch, tracks). The interaction model in this system is context dependent as different actions require different levels of system initiative. For example, if the user were to issue the query "make the drum track volume lower" the system would lower the volume in any track identidied as a drum track. This is analogue to the interaction modality seen in WATER4-NL where a command is issued and the system reacts to it. On the other hand, the user in this music generator could also issue a query like "add a piano track to this song" which requires a higher level of initiative by the generative components of the system. In response to such a query, the system

could present the user with a set of solutions that can be iteratively explored using an interactive evolution modality such as the track generator example from this section. This way the user can choose from a variety of initial solutions that can guide them towards their objective using the tool. This system showcases how natural language inputs can be used to control different system components that requiere different levels of initiative. Figure 4.11 shows a sample interaction session for this system.

4.6 Conclusions

In this paper we introduced and discussed a methodology to allow PCG systems designers to allow natural language input in their systems. This methodology follows a series of steps that allows a mapping of parameters in the design space of the PCG system to a series of keywords that are descriptive of the generative artifact and which allow the user to express design space movements in natural language form. This is accomplished by: (a) an initial exploration of thedesign space to understand the functionality of each parameter, (b) the creation of a design vocabulary, (c) a set of descriptive keywords related to the artifact's characteristics and a set of example design points related to each word in the design vocabulary, followed by (d) an expansion of the design vocabulary through the usage of synonyms and counter-examples that will form a conceptual feature. Afterwards, we design a series of natural language query patterns that map actions

in our generator to keywords and conceptual features. Furthermore, we discuss the implementation of a model to translate our natural language mappings to numerical movements in the design space. In addition, we provide an example implementation of our methodology to an existing plug-in in Unity3D called Water4[56]. Finally, we showcase a set of three theoretical applications of this methodology that exhibit different levels of initiative from the generator's side.

We believe that using natural language-based interfaces provides designers a way to let less technically oriented users explore the design space of a PCG system. By allowing the usage of expressive language, users can explore the design space of a PCG system in terms of the desired characteristics of the artifact, rather than a direct freeform manipulation of the parameters that compose the design space of the system. The affordances allowed by natural language descriptions of the artifact and the usage of conceptual features also allow for a reduction of interface complexity of mixed-initiative PCG systems. We believe that this methodology and its applications can improve the user experience of PCG systems, as well as providing a further reduction of the technical barrier of usage for systems.

Chapter 5

Evaluating Natural Language Interfaces For Mixed-Initiative Game Design Tools

5.1 Introduction

In the previous two chapters, we introduced two different natural language interface based systems for PCG. CADI [30] (a simple AGD system that generates variations of the game Pong) and WATER4-NL (a shader manipulation interface for the Water4 plugin in Unity3D). These two systems show how a natural language interface can be used by those with a lower amount of technical knowledge to explore an ample, high dimensional design space by using natural language

queries. These queries allow the user to express the desired characteristics of the generator's output in terms that reflect their intent. Queries such as "make the ocean angrier" capture the intent of the designer to modify a set of parameters that reflect the desired quality of the artifact without having to manipulate it directly by finding the parameters on a traditional GUI and then modifying the values one by one. Also, this method of input allows for a simplification of the user interface, as well. Compared to traditional graphical user interface based systems where the user is presented with one control (slider, knob, text boxes...) per parameter in the design space, natural language interfaces reduce input to either text or voice, with the option to present the user with appropriate graphical controls for the features selected by the natural language interface to modify. Based on these aspects, we believe that natural language interfaces provide a new way to interact with procedural content generators in such a way that users with non-technical backgrounds can explore complex design spaces by using expressive natural language descriptions of the aspects of the generator's output they want to modify.

To validate our proposition, we performed a user study that compares two different interfaces for the Water4 plugin in Unity3D. In this study, we want to explore how a user interacts with a procedural content generator using two different user interface modalities (traditional GUI and natural language). The subjects were tasked to draw a version of the artifact. The artifact was a representation

of the ocean that they want to create with Water4. Along with the drawing, the subjects provide a textual description of the picture. The drawing will serve as the reference point in the design space that they are trying to reach using the tool (Water4). Once the users have created their reference drawings and description, they are presented with one of the two user interfaces to build their reference artifact. Once they complete the artifact to their satisfaction, they are given a second user interface and are asked to repeat the same task. Finally, the subject will answer a survey about their user interface preferences.

We made this user study for us to be able to evaluate whether natural language input translates to an experience that reduces both the complexity of Water4's user interface, but also allows users to explore the generator's design space effectively using natural language queries. The study in this chapter tries to answer the following research questions:

1. **Do natural language interfaces reduce interface complexity for a complex high dimensional design space such as Water4?** We can reduce the number of controls presented at once to the user with a natural language interface. However, we want to know whether the reduction in interface complexity results in better interaction with the generator.

2. **Can natural language interfaces capture the design intent of the user and result in meaningful interactions?** We want to test whether the methodology detailed in the previous chapter can result in meaningful

interactions with the generator that allows users to use natural language descriptions of the output to explore the design space.

3. **What emergent issues arise using natural language input to interact with a procedural content generator?** We want to understand what interaction issues arise when users interact with Water4-NL compared to a traditional GUI, why these issues happen, and how we can address them to improve the user experience of our systems.

By answering these questions using our study, we can understand the advantages and disadvantages inherent to implementing natural language interfaces in procedural content generation systems. The insights derived from this study can be used to further improve our design methodology and the implementation of natural language interfaces within the context of game design tools. The rest of the chapter covers the methodology used to execute the user study in detail, followed by a discussion of the results and observations in this study.

5.2 Methodology

We divided the user study into three different phases. First is the design and preparation phase, where the subject creates a visual and verbal representation of the generator output they want to obtain. Task execution is the next phase, where the subject creates the artifact designed in the first phase using Water4's

native UI and with WATER4-NL. Finally, in the survey phase the subject answers a series of questions about the user experience of the two presented interfaces.

5.2.1 Design and Preparation Phase

The first phase in this study is the design and preparation of the subject for the task execution phase. We present the user with a description of the study and provide instructions for the first task. In this part of this phase, we tell the subject they will interact with a design tool that creates animated bodies of water in 3D, and we show a couple of examples of the generator's output to have an idea of what artifacts the system produces.

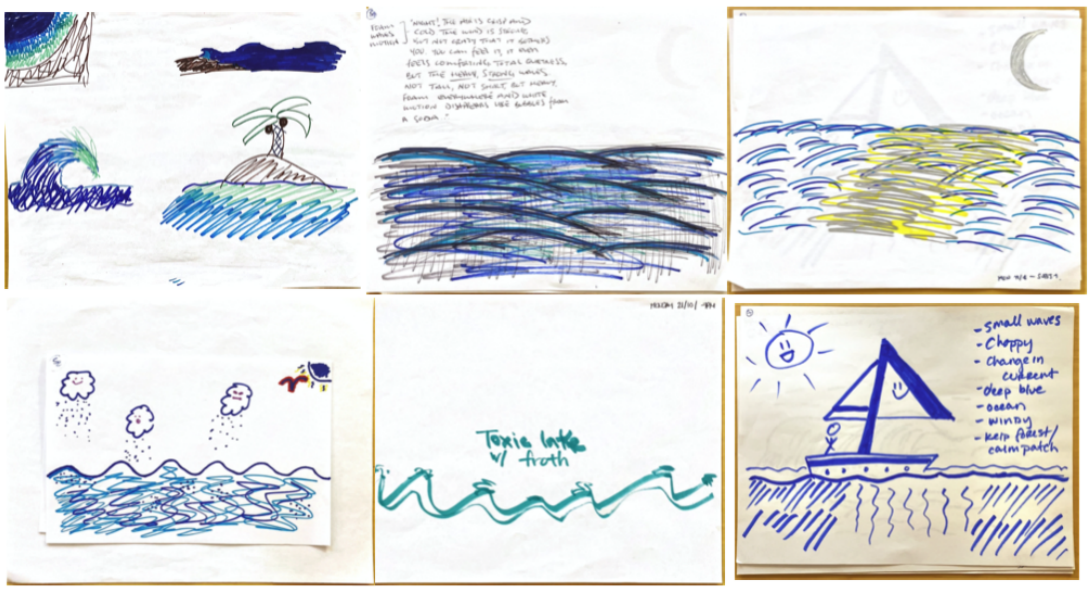


Figure 5.1: A sample of six different drawings with text descriptions of the ocean.

After we explain the task, the subject is told to create a drawing of the ocean,

preferably from a first-person facing-forward perspective. We also provide the subject with some informative questions that can help them guide the drawing process such as "Are the waves fast?", "Does the water have a light or dark color?", "Is the water clear", and "Is there foam in the water?". These questions can help the subject make a mental model of their intended artifact and help them draw their expected output. This helps address the "blank canvas" issue inherent to the design process.

Once the subject completes the drawing, they are asked to provide a textual description of the artifact. The subject is suggested to use simple declarative language (i.e., "the waves are tall," "the water is dark green") to describe the drawing. This textual description lets the subject create an initial vocabulary that describes what features in the generator they want to manipulate. Having a textual description of the expected output can help explore the design space using natural language commands, and to also have a "ballpark" of what parameters to look for in the graphical user interface. Figure 5.1 showcases a sample of the drawings and text descriptions made by different subjects.

We believe that this step in the study is a crucial component of our evaluation framework. Making the subjects in the study build a paper prototype of their expected output lets us get a reference point in the design space of the generator. It also allows us to replicate the experiment between different user interfaces. This phase of the study allows us to observe how users create a mental model

of the generator's design space by making a ballpark visual representation of the output they desire. Furthermore, textual description creates a vocabulary of what features in the design space need to be explored to reach the point in the design space that best represents their expected output. This paper prototype also allows users not to be confronted with a blank canvas during the execution phase. Were the subjects given the same task without this prototype they would face the blank canvas problem. They would both have to explore the design space and figure out what solution they want at the same time, which makes interaction with the generator difficult. Besides, having no "ground truth" can result in non-replicable design tasks for the two different user interfaces. This is because the subject's ideal design point representation might change without a reference as they complete the tasks.

5.2.2 Task Execution Phase

When the subject has finished building their paper prototype, they are asked to interact with the Water4 plugin in Unity3D. The subject has to create an artifact similar to the one in their paper prototype using two different interfaces (one traditional and one natural language). We describe the two interfaces in the experimental setup subsection in this chapter. Chapter 4 contains a detailed overview of the interfaces used in this user study. In this phase, the subject can ask the researcher questions about how to operate the tool or provide feedback

about the user experience. In order to avoid bias in our experimentation, we randomly choose the ordering of the user interfaces.

5.2.3 Survey Phase

Once the subject is done interacting with both user interfaces, we present them with a survey about their experience interacting with Water4 and WATER4-NL. In this survey, we ask the subject questions about the following: What user interfaces they prefer in terms of ease of use and complexity; questions about natural language interaction; and general feedback. The survey is structured as follows:

Questions about UI Preferences:

- From the two UIs presented to you in this study: Which one did you prefer the most to use in terms of how much they assisted you to achieve the task objective? (Rank A/B)
- From the two UIs presented to you in this study: Which one did you prefer the least to use in terms of how much they assisted you to achieve the task objective? (Rank A/B)
- For your most preferred UI: Describe what aspects of its design helped you achieve the task objective? (Text answer)

- For your most preferred UI: Describe what aspects of its design prevented you from achieving the task objective? (Text answer)
- For your least preferred UI: Describe what aspects of its design helped you achieve the task objective? (Text answer)
- For your least preferred UI: Describe what aspects of its design prevented you from achieving the task objective? (Text answer)

Questions about Natural Language UI Interactions:

- When using the natural language based UI: How much on a scale from 1 to 5 did you feel the UI listened to your commands? (Likert 1 to 5: 1 being not responsive, 5 being very responsive)
- Did you feel like you had to repeat the same command several times to achieve a desired effect when interacting with the tool? (Binary Yes/No)
- Did you try using the fine-tune controls (on the right side) to complete your task? (Binary Yes/No)
- If yes, did this help you avoid repetition of commands? (Binary Yes/No)
- Did you feel like there were problems with disambiguation when interacting with the UI? (Binary Yes/No)
- Please write any feedback about the natural language UI you want to provide to the researcher. (Text answer)

Questions about Interface Complexity

- From the UIs presented in this test: Which one did you perceive as the least complex to use? (Rank A/B)
- From the UIs presented in this test: Which one did you perceive as the most complex to use? (Rank A/B)
- For your most preferred UI: Describe the reasons for selecting it as the least complex? (Text answer)
- For your least preferred UI: Describe the reasons for selecting it as the most complex? (Text answer)

Questions about General Feedback

- Please leave any comments and feedback you want to provide to the researcher. (Text answer)

5.2.4 Experimental Setup

The experiment is set up is composed of two different user interfaces: One built-in into the Unity editor, as shown in figure 5.2, and one custom built by us running separately from the Unity3D editor. The first interface is the default editor user interface built for the Water4 plugin, and it has no modifications made to the plugin source code. The second interface was custom-built as a separate

application from the Unity3D editor. The details about how this user interface was designed and implemented can be found in chapter 4. We provided an initial starting point in the design space of the generator based on the default values for the plugin. This ensures that the task is reproducible across different user interfaces and different subjects.

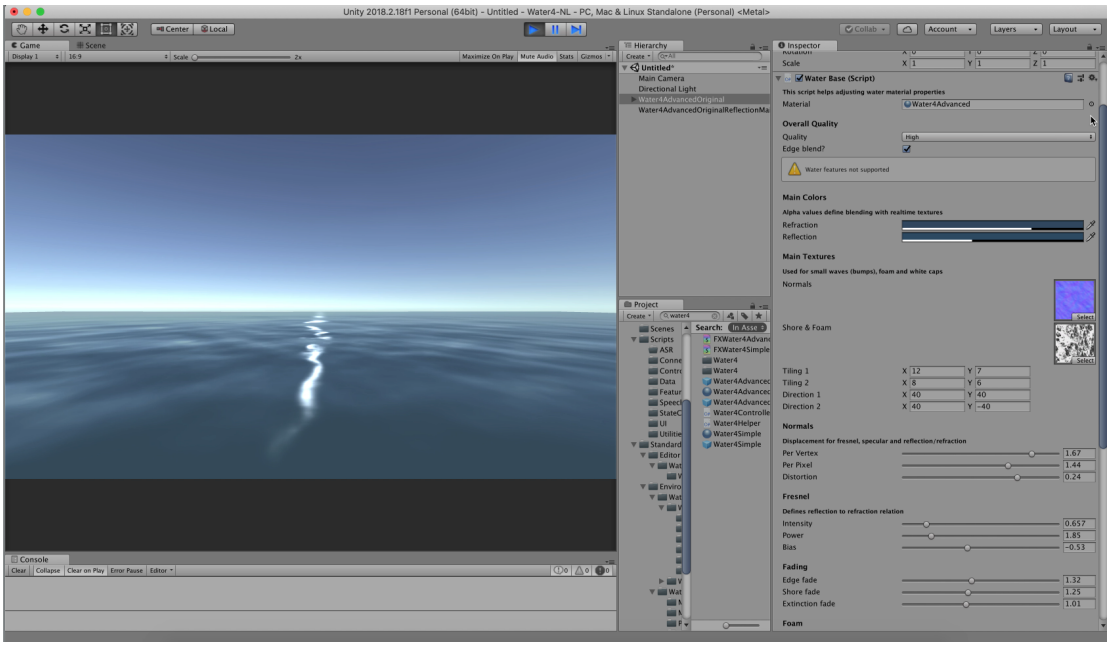


Figure 5.2: The stock Water4 UI within the Unity3D version 2019 editor..

5.2.5 Data Collection

We collected data from different sources during the implementation of this user study. The data used in this study comes from the following sources: The paper prototypes created in the first phase of the study, screen recordings of the user interacting with Unity3D, telemetry collected by our natural language interface,

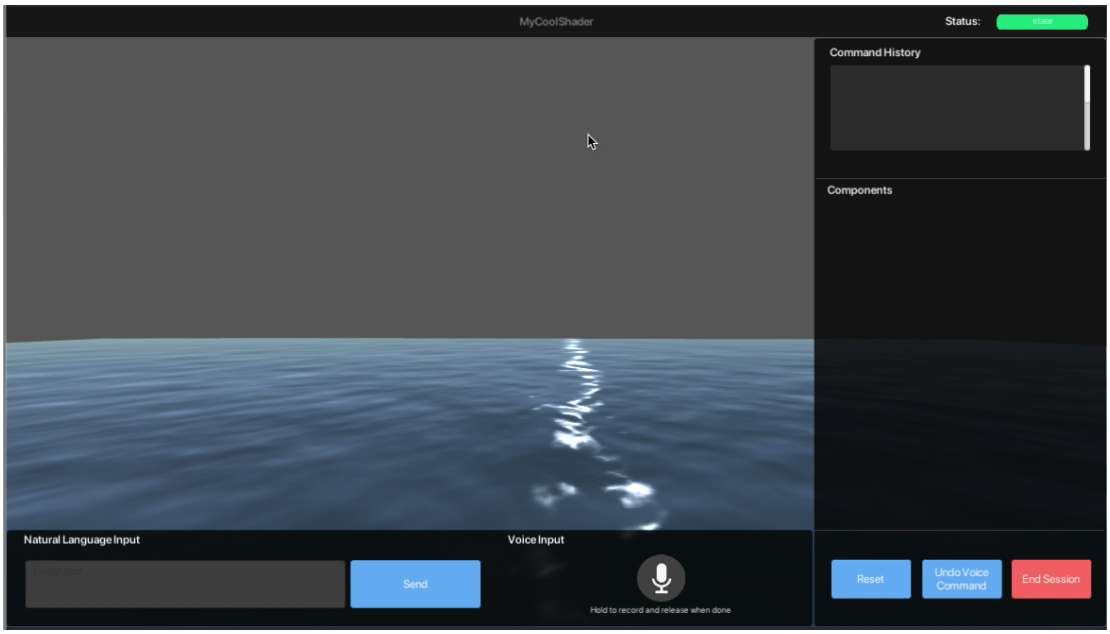


Figure 5.3: Our custom implementation of a natural language interface for Water4 called WATER4-NL.

notes taken by the researchers, and the survey given to the subjects at the end of the study.

Screen Recordings

We capture video of the computer screen of each subject while interacting with Unity3D using the traditional GUI (see Figure 5.2), or the natural language interface (see Figure 5.3). We annotated the recordings in order to extract insights about user actions with the different UIs.

Telemetry

The natural language interface contains a telemetry module that captures interaction data from the user. We store the captured actions in a JSON file that

is processed to provide a quantitative measure of how the user interacts with the natural language UI. Table 5.1 describes what data is collected by the telemetry module.

Name	Type	Description
time	datetime	Time when the action occurred
event_type	string	Type of action recorded
last_command	string	Top command in the command stack

Table 5.1: Description of the fields captured by the telemetry module in WATER4-NL

Note Taking

We collected notes during the observation of the subjects during the first and second phases of the study. These notes contain information about how the subject behaves, what affective reactions they have while using the interfaces, and verbal feedback provided to the researchers during the session.

5.2.6 Metrics

We define a series of metrics that can help us understand the effectiveness of each UI in our study in a quantitative manner. This subsection describes each metric, how it is measured, and an interpretation of the results. These metrics are measured from the telemetry data collected by WATER4-NL.

Listening Rate

We define the term "listening rate" as the number of voice/natural language queries perceived as understood to the user over the total number of queries made. An understood query is defined as a voice/natural language command given to WATER4-NL that resulted in the expected outcome for the user. This metric can be used to complement user affirmations of whether the system understood their design intent. The values range for this metric goes from 0 to 1. Zero, meaning that the system misunderstood all queries, and one meaning all queries were satisfactorily understood by the system.

Number of Undo/Reset Actions

We record how many errors the system commits regardless of the UI the user is testing. For this purpose, we implemented reset and undo commands to allow the user to go back or restart their design process while interacting with WATER4-NL. We define a failed action as a press of the undo button, which can be interpreted as a misunderstood query (see above) in a voice/natural language UI, or a misstep (wrong slider/unexpected effect from interacting with a control). A failed session can be defined by a press of the reset button, meaning that the resulting output by the system is far from the expectation of the user. The values for this metric can go from 0 to the total number of interactions with the system. Also, we can express in a 0 to 1 value range, similar to the listening rate metric, this can be

seen as an error/failed interaction rate for our user testing.

Time to Completion of Task

We measure the time a user completes each task in our study. Shorter times with a complete task can be interpreted as more successful sessions with WATER4-NL, whereas larger times might mean either that the user is exploring the system in greater depth or they are having difficulties with the tool. We plan to cross-check this metric with the number of interactions the user has with the system.

5.3 Study Results

We performed the study described in the previous section on a population of 10 local (San Francisco Bay Area) adults (defined as age 18 and over) with different technical backgrounds. Some of the users (n=4) had previous experience with Unity3D with the rest of the population having no experience with Unity but being comfortable with computational tasks like programming (n=2), and the rest having no experience with either.

5.3.1 Result Samples

We present a sample of the results collected from two of the user study subjects from the first and second phases of the study, as well as a transcript of the telemetry data collected showing the interactions they had with the natural language

interface.

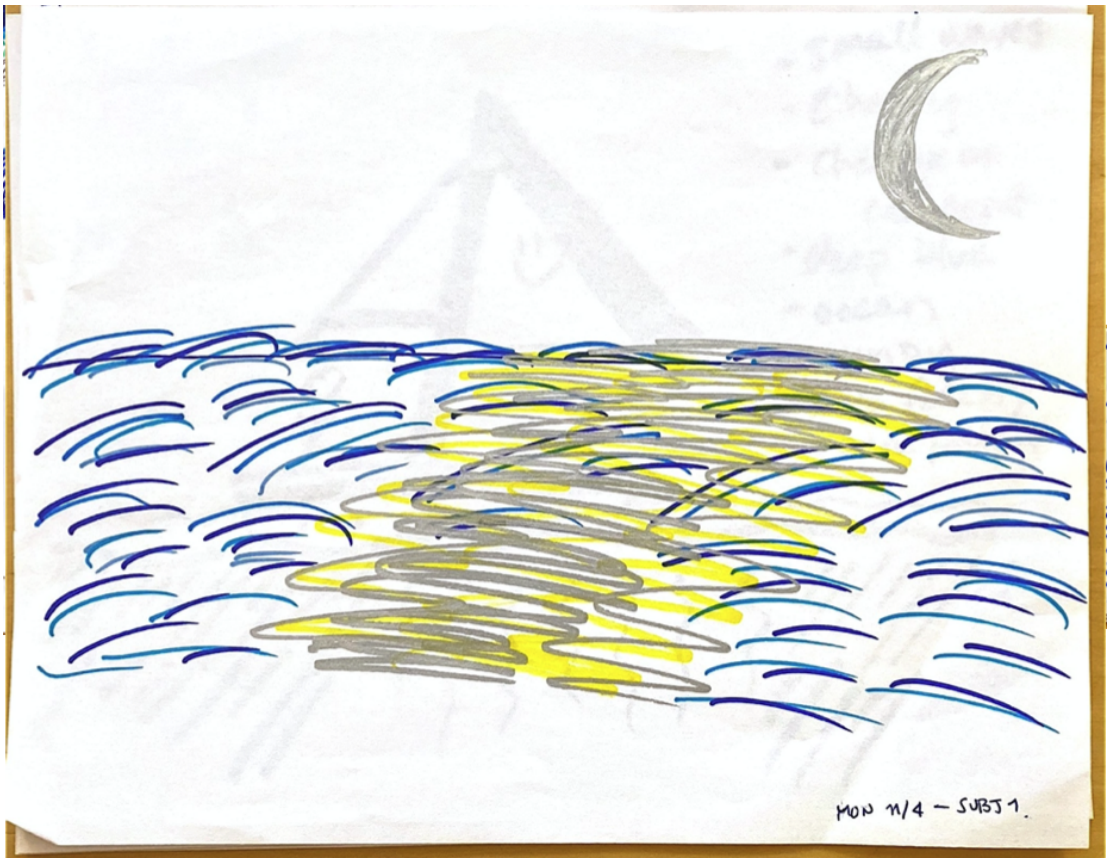


Figure 5.4: Sample initial drawing result from a user in the study.

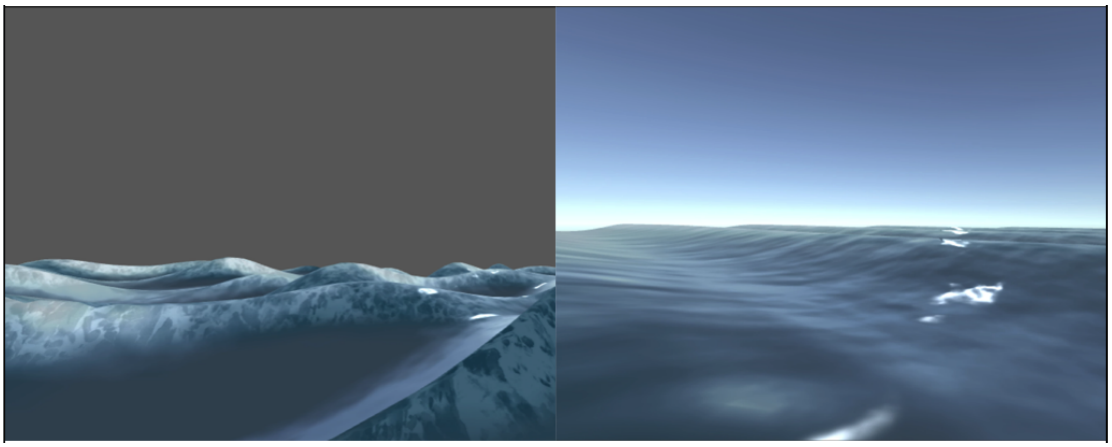


Figure 5.5: Sample task output from the NLI (left) and the GUI (right) for the first user in the study.

Time	Command
10/10/2019 20:40:34	*Start*
10/10/2019 20:40:39	*Start*
10/10/2019 20:41:11	*Make the waves taller*
10/10/2019 20:41:17	*Make the waves taller*
10/10/2019 20:41:45	*make the waves stronger*
10/10/2019 20:42:27	*make the water darker*
10/10/2019 20:43:02	*less foam*
10/10/2019 20:43:11	*make foam less intense*
10/10/2019 20:43:29	*make foam less intense*
10/10/2019 20:43:42	*make foam less intense*
10/10/2019 20:44:03	*make foam more turbulent*
10/10/2019 20:44:27	*make foam more turbulent*
10/10/2019 20:44:28	*make foam more turbulent*
10/10/2019 20:46:51	*make foam more turbulent*

Table 5.2: Sample telemetry log for the first user in the study.

Time	Command
11/22/2019 22:39:55	*Start*
11/22/2019 22:40:05	*add whitecaps*
11/22/2019 22:40:33	*make waves more frequent*
11/22/2019 22:40:38	*make waves more frequent*
11/22/2019 22:41:03	*shrinks away*
11/22/2019 22:41:10	*so yeah*
11/22/2019 22:41:46	*make waves smaller*
11/22/2019 22:42:03	*make waves faster*
11/22/2019 22:42:21	*make the water violent*
11/22/2019 22:42:30	*make the water violent*
11/22/2019 22:42:33	*make the water violent*
11/22/2019 22:42:38	*make the water violent*
11/22/2019 22:42:54	*make the water aggressive*
11/22/2019 22:43:53	*make the water calm*
11/22/2019 22:44:05	*make the water still*
11/22/2019 22:44:40	*add a title wave*
11/22/2019 22:45:02	*make the waves tall and wide*

Table 5.3: Sample telemetry log for the second user in the study.

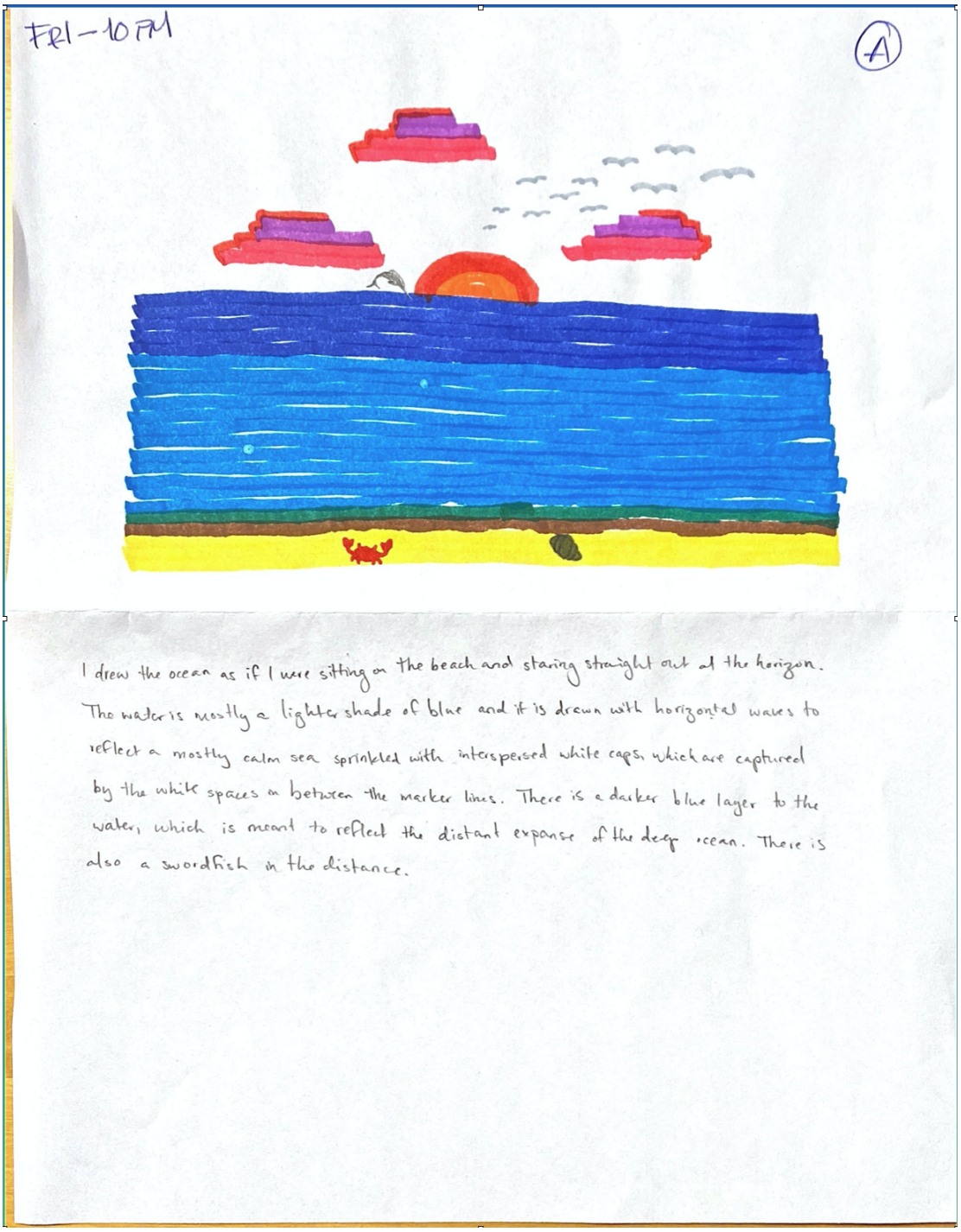


Figure 5.6: Sample initial drawing result from a second user in the study.

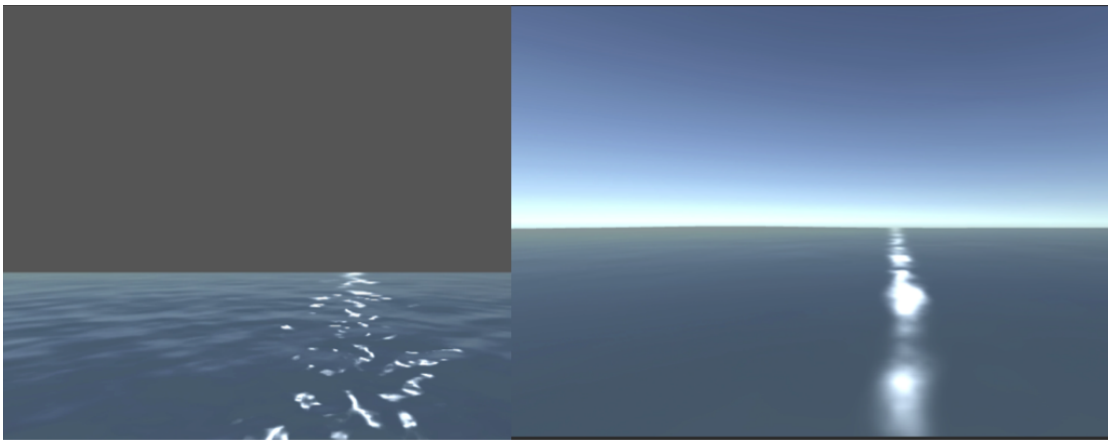


Figure 5.7: Sample task output from the NLI (left) and the GUI (right) for the second user in the study.

5.3.2 General Experience and Ease of Use

We asked our subjects several questions relating to the overall experience they had interacting with the two user interfaces presented to them in the study, as described by table 5.4 and figure 5.8. As the table illustrates, there is a slight preference among the subjects for the natural language interface in terms of ranking the UIs in terms of how much they assisted the subject in completing the task.

	Most Preferred UI	**Least Preferred UI**	**Total Population**
NLI	6	4	10
GUI	4	6	10

Table 5.4: Overall UI preferences reported by the subjects in the user study.

From the results of the survey, the subjects seem to prefer interacting with the natural language interface over the traditional UI. In order to validate our results for statistical significance, we performed a one-way Chi-Squared test on the results of this survey question. The test results gave us a p-value of 0.7518.

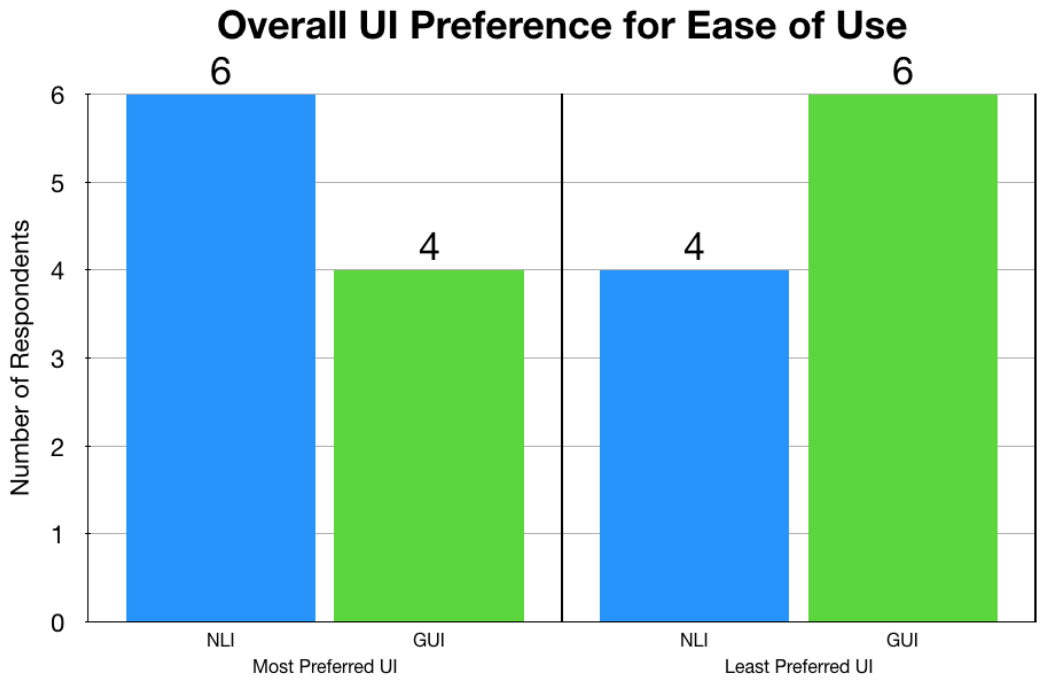


Figure 5.8: Charted results for the overall UI preferences in the study.

Based on this result, we determined that the result is not statistically significant. While these results are not conclusive, we decided to look at the qualitative data that we collected from the survey to support our hypothesis that users prefer interacting with natural language compared to a traditional UI. In addition to asking the users to rank their preferred UIs, we asked them what aspects of their most and least preferred UIs helped them complete the design task, and what aspects caused them difficulties while interacting with the UIs. We use the results of these questions to back our hypothesis.

When it comes to what aspects of the user interfaces presented helped the subjects achieve the task objective, subjects reported that using natural language

commands instead of manipulating graphical UI controls reduces the difficulty curve of interacting with Water4. The subjects also report that using natural language commands allows them to manipulate multiple parameters at the same time, compared to the one-by-one parameter space exploration process they experienced when using the traditional graphical UI. Finally, the subjects also report that the natural language allowed them to use common descriptive language that captured what the transformations they desired to apply to the artifact while interacting with the natural language UI. The list below shows some of the responses the subjects reported about their preferred aspects of the natural language UI:

- *"Since I didn't know the controls to get the effect that I wanted, I liked that I could just describe it and it would understand how I wanted the picture manipulated."*
- *"Deminiesed [sic.] the learning curve on what adjustments to make to get desired results."*
- *"Had the ability to change multiple values at the same time to get the most desired results.*
- *"using words to match to metrics for wave simulation"*

On the other hand, the subjects reported that the traditional graphical UI allowed them to complete the task faster because it presented all possible parameters at once, rather than issuing one natural language command at the time. In

this sense, these subjects preferred having more options to explore and modify rather than querying the system to present them (or modify) a subset of parameters selected by the natural language understanding component of our natural language UI. A sample of the subject responses regarding their preference of the traditional GUI are showcased in the following list.

- *"Speed, accuracy, ability to explore and learn the options"*
- *"I liked that there were so many parameters to try out, and choose the effects that I liked to keep."*
- *"GUI presented all of the possible parameters for manipulation at once, which made for a more explorative experience."*
- *"I knew exactly what I could interact with and could easily tell what they did since I could move the values to the extreme to see how it effected the water"*

Regarding the difficulties they encountered while interacting with the different UIs, the subjects reported that the natural language UI had problems with the interface sometimes not understanding their commands. This is because the subjects used words and patterns outside of the predefined vocabulary, which results in the system not modifying the artifact. The list below presents some of the comments left regarding the difficulties they encountered while interacting with the natural language interface.

- *"natural language controls don't detect some keywords"*

- *"when the words mapped to metrics that I did not understand. Especially since there were two and I did not know which one 'did the job'"*
- *"It might not have a bigger vocabulary of options to allow more designs implementation."*
- *"natural language controls don't detect some keywords"*

Regarding the traditional graphical UI, the subjects reported issues with not being able to understand what each control did. For example, some controls, when modified, provided little to no feedback to the user, whereas some controls did provide feedback to the user, but its naming was either confusing or too specific for a general user to understand. These concerns are shown in the list below.

- *"It wasn't always completely clear what it did and sometimes changing the values didn't have an obvious effect."*
- *"It would be nice to know more clearly what the different features would do to the animation"*
- *"I didn't understand at all what the different things did. Maybe if there was a scroll over tool that gave a description of what each thing did it would have helped. "*
- *"It took longer to play around with options and numbers to get the result I was looking for."*

5.3.3 Natural Language Interactions

We now look at the results of the study regarding natural language interactions. As seen in the previous section, we asked the users several questions about the interactions they had with the natural language interface, such as: whether they had to repeat commands, used the fine-tuning controls, and disambiguation. Table 5.5 and figure 5.9 show the results gathered from the user study survey.

N. Subjects	Had to repeat commands	Used fine-tuning controls	Fine-tuning helped avoid repetition	Problems with disambiguation
Yes	9	8	7	4
No	1	2	3	6

Table 5.5: Results for the natural language interaction questions in the user study survey.

As the results show, there was a vast majority of subjects in the study (N=9) that repeated the same command more than once consecutively to achieve the desired effect they wanted on the generated artifact. For example, if a user study wants to make really tall waves, it is possible that they might repeat the command "make the waves taller" several times consecutively to reach their expected destination in the design space. This is where the fine-tuning controls can help. In this implementation of the UI, we present the user the controls for the parameters that are being manipulated by the natural language interface every time a command is issued. This way, the user can further manipulate the appropriate parameters if the user is not completely satisfied with the solution presented by the natural language interface. A large number of these subjects (N=8) did use the fine-tuning controls to modify further the artifact they had to create for the task. Out of

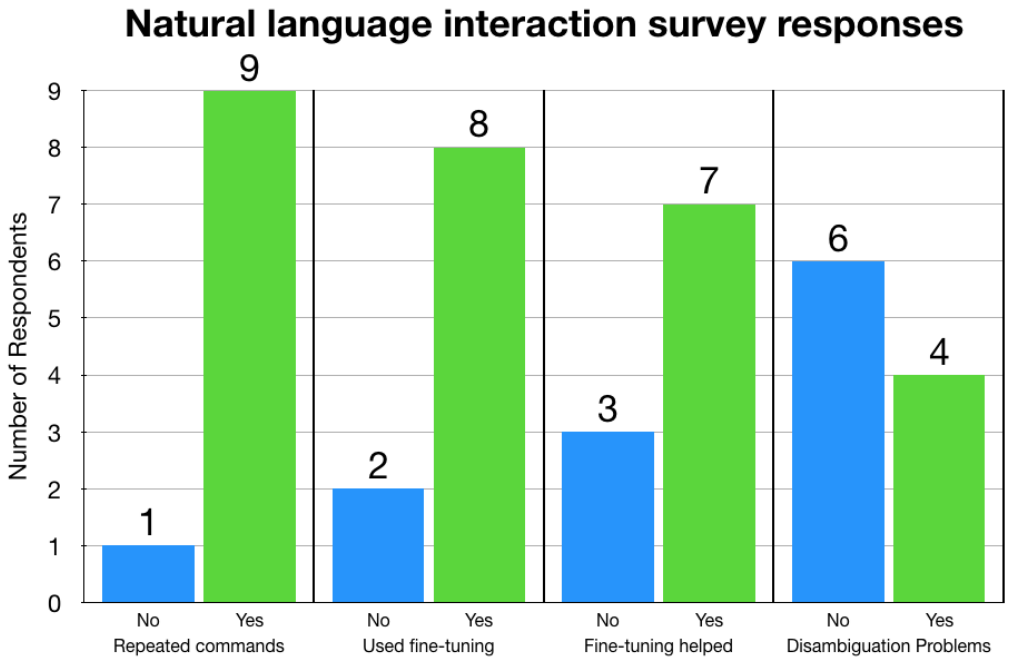


Figure 5.9: Charted results for the natural language interaction questions in the user study.

these subjects, the majority (N=7) reported that using the fine-tuning controls helped them avoid issues with command repetition. Regarding disambiguation issues, defined as receiving unexpected responses from a query, a slight majority (N=6) reported no problems with disambiguation. In order to supplement this self-reported data, we asked the subjects to note any observations they had about the interactions they had with the natural language interface. The following list shows some of the observations reported by the subjects:

- *"I thought it is a really cool idea to be able to talk to an animation and get*

it to do what is desired. While I defaulted to typing I think this version is especially interesting while speaking and might actually benefit from less control and only speaking to see how it is to interact with it that way and if users find it more or less enjoyable to exclusively use audio commands to modify the animation."

- *"I loved it. With reference to the question above about problems with disambiguation, I didn't have any. The only thing that happened sometimes was that the program didn't know what to do for a word that I used so it did nothing. It never did something the opposite of what I wanted or something that I didn't expect. It would be great to expand the vocabulary to include more words. Like Tumultuous which is a synonym for stormy. Some words convey a feeling. Since pictures convey feels in addition to just activities, including words that are often used to communicate the feeling that the picture gives you would be great. Like a peaceful meadow. Not sure how you would achieve that as I am not an artist but I would image that it would have something to do with the lighting and having the colors less vivid and slowing down any movement in the picture. Now I wish that I had tried that word to see if the ocean would have become calm. Excellent tool. I really loved playing with it. Like I said, I am not at all artistic but I think that for the water at least, I it helped me create a picture close to what I had envisioned. It would be great if the tools were expanded so that I could also*

change the colors and the shimmering on the water from the night sky. "

- *"Most of the commands I tried worked. But I had no idea how good the system would be to understand what I was saying. So I tried to keep the commands really simple, and use not very descriptive language"*

- *"I was scared that it wasn't going to recognise [sic.] emotions, but it was able to substitute emotions for other words (i.e. Anger = strength/speed)"*

These comments show that the subjects were able to interact with the natural language interface to achieve their goal to the best of their abilities without any major difficulties. The subjects, as seen in these comments, report that they had no problems with disambiguation, expressed surprise that emotional keywords were understood by the system, and found the interface to be mostly responsive to their commands.

In order to rate how responsive the natural language UI was to the subjects' commands, we asked them to rate the responsiveness of the natural language UI using a Likert scale ranging from 1 to 5. Table 5.6 and figure 5.10 illustrates the results of the subjects' ratings.

	Score				
	1 **Not Responsive at All**	**2** **Not Responsive**	**3** **Somewhat Responsive**	**4** **Responsive**	**5** **Very Responsive**
N. Subjects	0	0	3	7	0

Table 5.6: Results for the natural language interaction rating question about interface responsiveness.

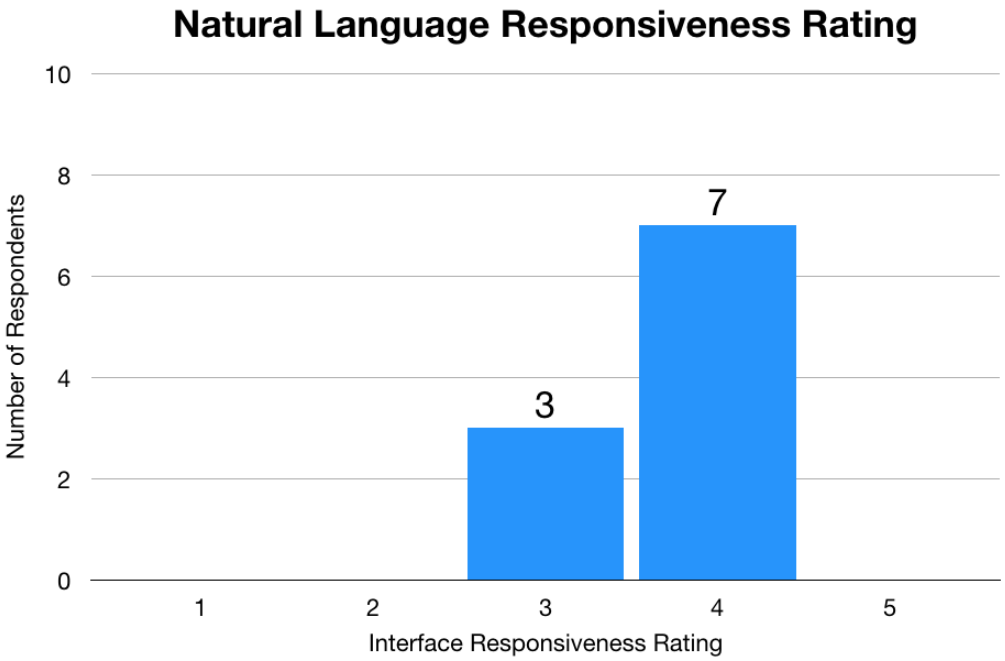


Figure 5.10: Charted results for the natural language responsiveness rating in the user study.

As seen in the table, the subjects in the user test found the natural language interface to be responsive enough to capture most of their desired queries. The average score across all subjects was reported to be 3.7 out of 5. This means that the users have found the natural language interface to respond to a majority of their commands while still missing specific patterns and keywords. This insight is complemented by the qualitative feedback left by the subjects in which they mention that they would like to have more query variations to use as seen in some of the comments below.

- *"Could be easier to say "more" or "less"*
- *"I feel like there might need to be a bigger word bank, but this might challenging too when thinking of words and phrases associated with the ocean and describing it."*
- *"many words for reflection were not detected. many words were mapped to metrics that I did not understand."*

For instance, one subject mentions that shorter query patterns could be useful to improve the user experience of the natural language interface. In this case, the subject mentions that having query patterns that go beyond "make the X more/less Y" such as "add X to the scene" or "change the Y of X" can help with the responsiveness of the natural language interface. In addition, several subjects report having problems with the recognition of specific keywords, which can be solved by expanding the vocabulary of our natural language comparison. Further down in this chapter we will compare these self-reported ratings for the natural language UI to our telemetry results to gain further insights about the responsiveness of the interface.

5.3.4 Interface Complexity

The last section of the survey asked the subjects to rank the user interfaces according to their complexity. Interface complexity is defined in this study as a

combination of different factors related to the composition of the user interface. Some of these factors are: The number of controls displayed at once to the user, understanding of information, and navigability. Table 5.7 and figure 5.11 show the results of this section of the survey.

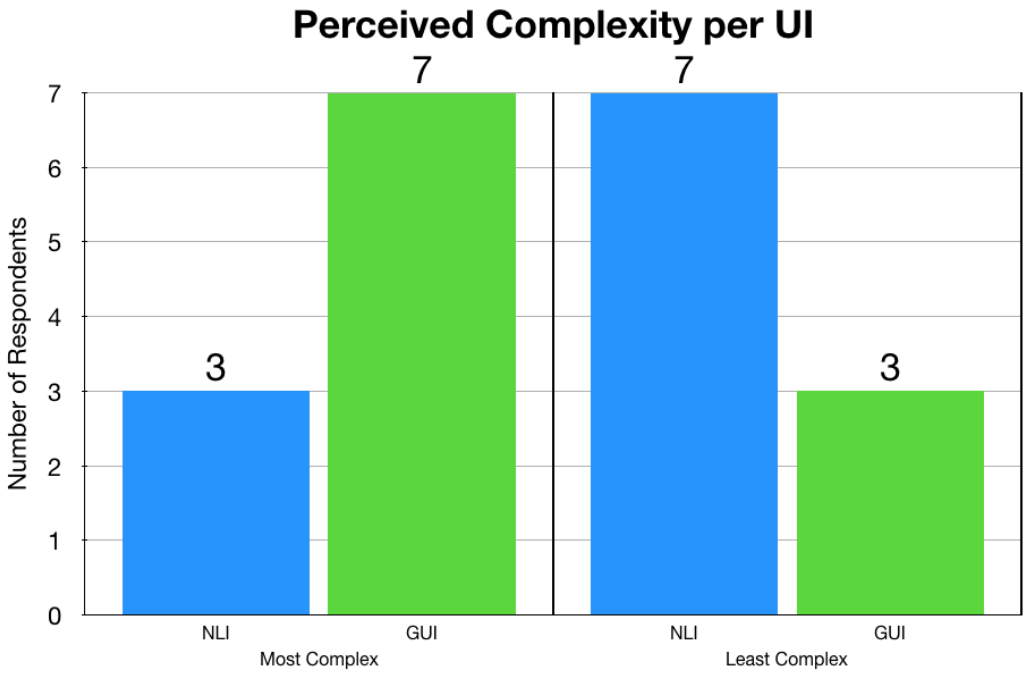


Figure 5.11: Charted results for the natural language interaction questions in the user study.

	Most Complex UI	Least Complex UI	Total Population
NLI	3	7	10
GUI	7	3	10

Table 5.7: Overall UI preferences ranked by complexity reported by the subjects in the user study.

The results show that a majority of users (N=7) perceive the natural language interface as less complex than the traditional graphical UI. Compared to the data

for overall preference and ease of use, a slightly larger amount of subjects (including one subject that preferred the traditional GUI overall) acknowledge that the natural language interface is less complex. To test for statistical significance we performed a one-way Chi-square test over our hypothesis that more subjects would find the natural language interface as less complex. The test resulted in a p-value of 0.3428 which means the result is statistically insignificant. Given this result, we decided to look at the qualitative data provided by the questions in this section of the survey to support our hypothesis.

We asked the subjects to mention what aspects of their preferred UI made them perceive it as less complex. The subjects that preferred the natural language interface commented that being able to use descriptive commands reduced the complexity of the interface. This is because they mention that using language helped them think less about the specifics of the design space and instead focus on the larger picture of what they wanted to achieve in the design task. The list below showcases some of the answers provided by the subject in support of the natural language interface.

- *"It was self explanatory really, just say a command. Just write things"*
- *"I didn't have to worry about knowing what values to put where to get desired results"*
- *"There's less controls (which then respond better)"*

- *"NLI was much easier to use because I didn't have to figure anything out. I could just say what I wanted and then the program knew which things needed to be changed to get the result that I wanted. Super easy."*

In contrast, the users that prefer the traditional GUI comment that having a more substantial amount of controls to choose made them feel more comfortable exploring the design space. The following comments show a sample of what the users commented respect to their preference:

- *"Everything you could change was in the bar, so I wasn't unsure if I was missing something. Also, I was only interacting with sliders/numbers instead of going back and forth between the text bar and the sliders."*

- *"I liked that there were so many parameters to try out, and choose the effects that I liked to keep."*

- *"I knew how to control the changeable parameters, because the commands were descriptive."*

Finally, we asked the subjects to mention what aspects of their least preferred UI made them perceive it as more complex. Subjects who perceived the natural language UI as more complex say that having to use descriptive commands linearly made it more complex to use, as the user had "to strategize to figure out the phrasing and order of commands" to get whatever parameter they wanted to modify. Other comments mention that navigating between the text/voice input in

the natural language UI and the fine-tune controls made the interface complicated to use, and that the interface could be "more efficient" and that it was "challenging to learn". The following sample answers from the survey showcase these concerns made by the subjects.

- *"Going back and forth between the text bar and the sliders, and the sliders going away and having to retype commands made it more complicated"*
- *"Could be more efficient overall, more challenging to learn but seems more powerful"*
- *"Every manipulation to the water had to be entered as a separate command, so it took some strategizing to figure out how to phrase and order the commands in order to get certain parameters to show up"*

On the other hand, the subjects that perceived the traditional GUI as more complex mention that there were too many controls to choose from, that the parameter names were difficult to understand, and that some parameter values did not provide feedback as expected. This concerns are exemplified by the comments below:

- *"[The] GUI still had a ton of information and options. The terminology was extensive and overwhelming compared to describing my image to a computer."*

- *Confusing on what values where going to change what I wanted to change.“ didn't understand at all what the different things did. Maybe if there was a scroll over tool that gave a description of what each thing did it would have helped. ”*

- *“I had to look for metrics that made a difference. Then I had to understand what they meant by experimenting which took more time than the first.”*

- *“It had so many sliders and fill in the box numbers with no explanation about what each of them actually did. I think it would take hours to learn all of the various things that can be manipulated and how they interact with each other to affect the picture. “*

5.3.5 Telemetry and Metrics

In addition to collecting user feedback from the survey, we collected telemetry data from the users while interacting with WATER4-NL and made screen recordings of the user interacting with Water4 and WATER4-NL. Section 5.2 details how the data was collected and what information was captured from the user. We implemented the telemetry and recorded the user study sections to paint a quantitative picture of how the subjects interacted with the different user interfaces to complement the insights we gained from the qualitative data collected from the survey.

Time To Completion of Task

One of the metrics that can help us gain a quick insight at how effective an user interface is for a task is how much time it takes to complete it. In this case, using the video recording, we were able to capture the time it took each user to complete each task. Table 5.8 shows the statistics calculated from the video recording timestamps.

UI Type	Average Time (secs)	Min Time (secs)	Max Time (secs)
GUI	590.9	305	902
NLI	470	322	1070

Table 5.8: Times to completion of task across all users for the two interfaces tested in the study.

As the table suggests, the users were capable of completing the task faster on average using the natural language interface compared to the traditional graphical UI. In this case, the average reported difference to complete the tasks is 120.9 seconds. To test for statistical validity of our hypothesis that the subjects were able to complete the task faster with the natural language interface we performed a t-test. The test resulted in a p-value of 0.152 which means the result is not statistically significant. To gain a better understanding and support our hypothesis, we decided to look at the task completion time differences per user and grouped them by their fastest session. Table 5.9 shows the time differences.

The data showed that only one subject completed the task faster using the traditional graphical UI. In this case, the subject took 12 minutes and 45 seconds

UI Type	Average Diff (secs)	Min Diff (secs)	Max Diff (secs)
GUI	765	765	765
NLI	246.8	94	518

Table 5.9: Difference of task completions across all users for the grouped by interfaces tested in the study.

more to complete the task using the natural language interface. With this outlier removed, the subjects who completed the task faster with the natural language interface took on average 4 minutes and 6 seconds less to finish the task compared to the traditional GUI. Measuring the difference between the completion times per-interface compared to an average of the time to complete a task provides us with a clearer picture of how effective each user interface is. Having gained this insight we removed the outlier and recalculated the time to completion task metrics. Table 5.10 shows the recomputed statistics for this metric.

UI Type	Average Time (secs)	Min Time (secs)	Max Time (secs)
GUI	631.7	465	902
NLI	384.86	322	492

Table 5.10: Times to completion of task across all users for the two interfaces tested in the study.

The recomputed metric calculations show a more optimistic picture. In this case, the remaining subjects were able to complete the task with the natural language interface faster by an average of 246.85 seconds. We tested the statistical significance of our hypothesis using the same test. The results in this case were more encouraging as the t-test yielded a p-value of 0.00218 which can be interpreted as statistically significant. All compounded, the results for this metric

are highly encouraging as they show a quantitatively measured difference in task completion times in favor of the natural language interface.

Number of Undo/Reset Actions

Another metric that we are interested in is the number of undo and reset actions each subject performed when interacting with the natural language interface. We collected information about when the users pressed the reset or undo button in WATER4-NL. We interpret an undo or reset action after a text command as a potential misstep from the natural language interface, such as modifying the wrong parameter or moving too much or too little in the design space. In this study, only one subject performed undo and reset actions across all interaction sessions. This subject performed two reset actions, and four undo actions. Given the small amount of data, no generalizable insights can be derived from this metric.

Listening Rate

The last metric we measured was the listening rate of the interface. We defined the listening rate of the interface as the number of accepted commands by the NLU component of WATER4-NL over the total number of commands issued by a subject. Table 5.11 shows the results of our computed listening rates.

Average Rate	Min Rate	Max Rate
0.7	0.47	1

Table 5.11: Recorded listening rates for the natural language interface used in the study.

The table shows that on average 70 percent of the commands were accepted by WATER4-NL's NLU component. This result seems to be consistent with the subjects' qualitative feedback. Compared to their self reported responsiveness rating for the natural language interface of 3.7 out of 5, which meant that the users found the interface to be responsive enough, but not enough to understand their every command. The calculated listening rate seems to correspond with what the subjects reported. This quantitative metric is also consistent with the qualitative data collected in the survey. The subjects reported that the system sometimes did not understand certain keywords or patterns. In order to gain a better understanding of WATER4-NL's calculated listening rate we analyzed what commands and keywords were causing the most missed interactions. Table 5.12 shows the top 5 accepted commands, and the top 5 missed commands captured by our telemetry data.

Accepted Command	**Times Recorded**
make the water fast	7
make the waves faster	5
make the waves more frequent	4
make the water slower	3
make the water darker	3

Missed Command	**Times Recorded**
make the water violent	4
make the water brighter	3
make the water choppy	2
make the water tense	2
make the ocean belligerent	1

Table 5.12: Top 5 accepted commands (left) and missed commands (right) issued by the subjects to the natural language interface.

The table shows that the top missed commands are synonyms of keywords captured by our NLU component. For example, "make the water violent" can be replaced by "make the water angry" which is a valid command for the NLU component. This is consistent with our qualitative data, in which the subjects

mention that certain keywords are not being recognized. It is interesting to note, that the users ended up finding acceptable commands, such as "make the water lighter" instead of "make the water brighter" during the interaction sessions in the study.

5.4 Discussion

In this study, we wanted to answer three research questions about the user experience of procedural content generation tools: Can natural language input reduce interface complexity? Can we capture designer intent and create meaningful interactions using natural language? And, what issues emerge from using natural language input in procedural content generation?

With regards to interface complexity, the user study shows that the subjects perceived the natural language interface as less complex. The cause of this perception is that presenting a lower amount of interactable controls, which also have "humanized" names is seen as simpler in the eyes of the subjects. The qualitative data tells us that the subjects that think natural language interfaces are less complex prefer using a higher level abstraction for their input (thinking about characteristics of the artifact) over the high specificity of the graphical user interface for Unity's Water4 (having to learn what each parameter does and why). In this sense, natural language input allows for a reduction of the amount of information presented to the user, but also provides them with a means to not have to learn

the specifics of the design space. By using natural language queries such as "make the water faster", or "make the waves taller" the user can not think about what lies in the parameter space. For example, Water4 uses a Gerstner wave simulation to animate the shader and the traditional UI presents the parameters named by their technical descriptions (Gerstner Amplitude, Gerstner Speed). This presents a challenge to the user as they have to learn the technicisms associated with the generator's design space. In contrast, the natural language UI allows users to express the desired characteristics in descriptive terms without having to learn the specific parameter names in the design space. Based on the results of the study we feel encouraged by the subjects' feedback and overall preferences.

When it comes to natural language interactions, the subjects reported having an enjoyable experience interacting with WATER4-NL. Based on both telemetry and self-reported quantitative data the subjects perceived the interface to be fairly responsive to their commands. The self reported responsivity rating of 3.75 out of 5 by the subjects complemented by the 0.7 (70 percent) listening rate calculated from our telemetry logs shows that the users can effectively express their intent to the generator but that not all commands are understood. In addition, the vast majority of users reported to have repeated commands to achieve the desired effect they wanted from the command. While the fine-tune controls helped with avoiding repetition eventually, there is an area of opportunity to better capture the notions of magnitude in natural language commands. Regarding designer intent,

the qualitative data showed that the subjects were able to use emotional language about the artifact (for example: "make the waves angry", "make the ocean calm") and that this type of descriptive commands allowed them to not think about the technical aspects of modifying the artifact. This can be seen as an encouraging sign about the notion of capturing designer intent with natural language. By allowing the users to explore the design space with natural language we create a new layer of abstraction that lets the users not think about the technicalities of the artifact and think in bigger picture terms related to their overall design task. In this sense, we create a mental model of the generator's design space through our vocabulary and query patterns. This mental model is able to capture the users' design intent by categorizing the parameter space into their natural language descriptors.

Finally, while the results are encouraging there are avenues for improving our natural language interface. The main issue that the subjects reported was the lack of variety of accepted keywords and natural language query patterns. This issue can be addressed by either expanding the vocabulary (manually, or via data-driven techniques) and building a larger amount of query patterns beyond "make X more/less Y". All in all we are encouraged by the results, feedback, and insights gained by this evaluation.

5.5 Conclusions

In this chapter we evaluated the effectiveness of natural language interfaces for procedural content generation. We performed a study in which the subjects were tasked with recreating a drawn description of the ocean using the Water4 plugin for Unity3D. The users in the study interacted with two different interfaces, a natural language interface called WATER4-NL built by us, and the default user interface provided by the engine. After the users completed their design tasks, they were asked to answer a survey related to their experience interacting with the interfaces. The survey covered aspects such as interface complexity, natural language interactions, and ease of use. In addition, telemetry data was recorded from the subjects' interaction with WATER4-NL to have a quantitative measure of how the subjects interacted with WATER4-NL. The results of the study show that the subjects perceived the natural language interface as less complex, fairly responsive to their commands, able to capture their design intent and generally faster to use than the traditional graphical UI for Water4. Nonetheless, the subjects noted that there could be a larger vocabulary and a wider variety of natural language query patterns to improve their experience with the natural language UI.

We believe that this evaluation provided us with answers to whether natural language input can capture designer intent and reduce interface complexity for procedural content generation. While the users did perceive natural language

interfaces as less complex and had a good experience overall interacting through this method, there is room for improvement particularly when it comes to the vocabulary and query pattern range.

Chapter 6

Conclusion and Future Work

6.1 Summary

In this work, we discuss the new creative support modalities that natural language interfaces provide in procedural content generation systems. We believe that natural language input can overcome some of the issues present in procedural content generators that have large and complex design spaces such as interface complexity and the lack of a model of designer intent. By reducing input to only natural language (either voice or text), we eliminate the large amount of GUI controls that are expected from a complex design space. In addition, using natural language provides users with an expressive means of input that captures their intent.

First, we offer a survey of the literature that informed the implementation of

the works produced in this dissertation. After the literature review, we introduce CADI a first experiment in the development of natural language interfaces for procedural content generation. This system takes natural language input from either voice or text and converts it into movements within a parameterized fluidic game-like design space of variations of the game Pong. Building CADI provided us a set of first explorations on how we can map natural language concepts to a set of parameter values in complex multi-faceted design spaces that is common in automated design systems.

In the next chapter, we present a methodology for designing and implementing natural language interfaces. This methodology covers the essential steps required by a technical designer or researcher to create meaningful natural language interactions for their procedural content generation systems such as: Creating a functional mapping of the design space, setting an initial descriptive vocabulary of the output, expanding the vocabulary and creating multi-parameter conceptual features, and designing natural language query patterns. This chapter also provides an example of a natural language interface built for the Water4 plugin of the Unity3D engine called WATER4-NL. We implemented this interface to showcase how we can apply the methodology to an already existing system. WATER4-NL allows users to navigate the parametric design space of an animated water shader using either text or voice queries related to the artifact.

The last chapter covers an evaluation of natural language interfaces in pro-

cedural content generation systems through a user study. In this user study, we presented the subjects with the task of interacting with two different user interfaces in Unity3D to re-create a drawn representation of the ocean's water using the Water4 plugin. The study was divided into three phases: In the first phase, the subjects make a drawing of the ocean along with a text description of their drawing. This serves as a reference point for them to re-create using the two different user interfaces. In the second phase, the subjects are tasked to re-create their drawings using Water4. During this phase, the subjects are presented with the traditional GUI and our implementation, WATER4-NL. Once the subjects are content with their created artifacts, they move on to the next phase of the study. Finally, the subjects answer a survey asking about their experience with the following aspects of the user interfaces that they were presented. These aspects are ease of use, natural language interactions, and interface complexity.

Our results show that when it comes to ease of use, users had a slight preference for the natural language interface over the traditional GUI. The subjects mention that they prefer the convenience of using descriptive language to manipulate the parameters in the design space compared to learning what each control in the GUI does. Their main issue with the natural language interface were related to the number of queries the system understood, meaning that there needs to be further work to be done to expand the number of possible patterns that a user can express when using the natural language interface. When it comes to natural language

interactions, the subjects report that the interface to be fairly responsive to their commands but once again had issues with the variety of natural language queries the system accepted.

In addition, the subjects report having to repeat a command several times to achieve the desired effect of their intended natural language query, although having fine-tuning graphical controls presented to them helped them avoid repetition. Finally, the majority of subjects found the natural language interface to be less complex than the traditional GUI. In this aspect, the subjects mention preferring the natural language interface because of their lower difficulty curve of use, use of descriptive language, and increased speed of use. In comparison, the traditional GUI was seen as confusing, and with too many hard to understand controls.

The results of the study show the potential natural language interfaces have to provide a simplified user interface for procedural content generation that allows users to interact with a complex design space using descriptive natural language queries. While the users preferred natural language interactions, there are areas of improvement when it comes to the design of the natural language query patterns. The results showed that there is a need for an expanded set of natural language commands that the user can issue to the system. All in all, we believe that the results of this user study provide an argument about the usefulness of new modes of interaction in procedural content generation.

6.2 Future Work

The work in this dissertation is only the beginning in the exploration of the new modalities of creative support that natural language interfaces provide in the field of procedural content generation. We believe that integrating natural language input capabilities into procedural content generation systems, and into game design tools can lower the technical barriers of interactive content creation. One of the areas of future work in this direction is the improvement of the understanding capabilities of natural language interfaces. As our study showed, the main issue found when interacting with a natural language interface was the lack of variety of query patterns that can be used.

In regards to improvements, the easiest step forward would involve authoring more patterns as testing continues. That being said, one could imagine several avenues forward towards developing a better generalizable set of query patterns. For example, crowdsourcing can be used to collect data to create a corpus of commands and patterns that are common to a variety of game design tasks by creating very simple procedural generation casual creator [12] type tools to collect data. This data can then be used as labeled training data for a machine learning-based NLU component in place of a pattern matching system such as Chatscript. Furthermore, this crowdsourced corpus can be used as a basis for common natural language queries for common operations in game design tools. Examples of this are: graphics transformations such as rotation, scale, and posi-

tioning, instantiating of common game entities like NPCs or items, and standard animation operations such as playback speed. This way, with some integration work, natural language capabilities could be added to more free-form game design tools outside the realm of procedural generation.

Natural language interaction in procedural generation is not limited to the conversational paradigm illustrated in CADI and WATER4-NL. We can use other forms of natural language input. For instance, one could imagine a system that takes a 1-3-5 format description of a game. 1-3-5 formats are common forms of initial brainstorming for game design in which a game is described in 1, 3, and 5 sentences, respectively. In this system, a game designer inputs a 1-3-5 description of their game. Then the recognized parts of speech (the character can jump, the game is set in space) are mapped into points within a parametric design space (agent actions, or physics parameters). This way, interdisciplinary game design teams can create quick prototypes that give them a ballpark of their idea. Another form of output could be taking the 1-3-5 description of a game and gathering the information that is helpful to the design team. For example, retrieving similar reference games like GameSage [47] does, collecting visual information in the form of a data-mined mood board. We believe integrating natural language input into the digital game development process can open up the process to a greater audience. This type of input allows non-technical designers to not think about technical details and instead focus on the bigger picture of the interactive experience they

desire to build.

It is also important to mention the importance of transparency and explainability as a future direction in this field of research. By showing the user the rationale of how the system interprets natural language commands is crucial to improve the user experience of natural language interfaces. Utilizing XAI techniques in future systems that use machine learning for natural language understanding can provide users with an understanding of what goes behind the scenes. This way, by presenting the user with an explanation of how the system understands the user's commands, we can improve our user experience. This can be in the form of correcting understanding mistakes, or by helping the user find alternative commands to achieve their goals.

6.3 Final Thoughts

The work in this thesis is only the tip of the iceberg when it comes to exploring the new creative support modalities that natural language input can bring to the development of procedural content generation systems. We believe that integrating natural language interfaces can help lower the difficulty of interacting with sophisticated procedural content generation algorithms with their lower interface complexity and models of designer intent. This work provides a set of contributions that will hopefully lead into more research about the user experience of procedural content generation tools, as well as the knowledge on how to de-

sign, implement, and evaluate natural language interfaces for procedural content generation.

Bibliography

[1] Allan Alcorn. Pong (game), 1972.

[2] A Baldwin, S Dahlskog, J M Font, and J Holmberg. Mixed-initiative procedural generation of dungeons using game design patterns. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 25–32, August 2017.

[3] Pippin Barr. Pongs (game), 2012.

[4] Gabriella A B Barros, Michael Cerny Green, Antonios Liapis, and Julian Togelius. Data-driven design: A case for maximalist game design. May 2018.

[5] C Browne and F Maire. Evolutionary game design. *IEEE Trans. Comput. Intell. AI Games*, 2(1):1–16, March 2010.

[6] Eric Butler, Adam M Smith, Yun-En Liu, and Zoran Popovic. A mixed-initiative tool for designing level progressions in games. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pages 377–386. ACM, October 2013.

[7] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Interactive evolution for the procedural generation of tracks in a high-end racing game. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 395–402, New York, NY, USA, 2011. ACM.

[8] Meghan Chandarana, Erica L Meszaros, Anna Trujillo, and B Danette Allen. Natural language based multimodal interface for UAV mission planning. *Proc. Hum. Fact. Ergon. Soc. Annu. Meet.*, 61(1):68–72, September 2017.

[9] Hyunmin Cheong, Wei Li, and Francesco Iorio. Automated extraction of system structure knowledge from text. page V02AT03A011, 08 2016.

[10] Simon Colton. Ml-based style transfer for game assets (video from the MetaMakers Institute ASYNC Online Conference oct '17), 2017.

[11] Simon Colton, Mark Nelson, Edward Powley, Swen Gaudl, Rob Saunders, Blanca Perez Ferrer, Peter Ivey, and Michael Cook. A Parameter-Space design methodology for casual creators. April 2018.

[12] Kate Compton and Michael Mateas. Casual creators. In *International Conference on Computational Creativity 2015*, pages 228–235, 2015.

[13] M Cook and S Colton. Multi-faceted evolution of simple arcade games. In *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, pages 289–296, August 2011.

[14] Michael Cook, Simon Colton, and Jeremy Gow. Automating game design in three dimensions. In *Proceedings of the AISB Symposium on AI and Games*, pages 20–24, 2014.

[15] Michael Cook, Simon Colton, and Alison Pease. Aesthetic considerations for automated platformer design. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.

[16] Samuel Cousins. Lahar (Software Package), 2017.

[17] Joe Decuir. Video olympics (game), 1977.

[18] Gary G. Hendrix. Natural-language interface. *American Journal of Computational Linguistics*, 8(2):56–61, 1982.

[19] Gary G Hendrix, Earl D Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. Developing a natural language interface to complex data. *ACM Trans. Database Syst.*, 3(2):105–147, June 1978.

[20] O J Hui, J Teo, and C K On. Interactive evolutionary programming for mobile games rules generation. In *2011 IEEE Conference on Sustainable Utilization and Development in Engineering and Technology (STUDENT)*, pages 95–100, October 2011.

[21] Amazon Inc. Alexa (assistant software), 2019.

[22] Apple Inc. Siri (software), 2019.

[23] Francesco Iorio, Hyunmin Cheong, Wei Li, L.H. Shu, Alex Tessier, and Erin Bradner. Natural language problem definition for computer-aided mechanical design. 04 2014.

[24] Aaron Isaksen, Dan Gopstein, Julian Togelius, and Andy Nealen. Discovering unique game variants. In *Computational Creativity and Games Workshop at the 2015 International Conference on Computational Creativity*, 2015.

[25] Vladimir Kulyukin. Human-Robot interaction through Gesture-Free spoken dialogue. *Autonomous Robots*, 16(3):239–257, May 2004.

[26] A Liapis, H P Martínez, J Togelius, and G N Yannakakis. Adaptive game level creation through rank-based interactive evolution. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pages 1–8, August 2013.

[27] A Liapis, G N Yannakakis, and Togelius J. Sentient sketchbook: Computer-aided game level authoring. In *Proceedings of the 8th Foundations of Digital Games Conference*, 2013.

[28] J C R Licklider. Man-Computer Symbiosis. *IRE Transactions on Human Factors in Electronics*, HFE-1(1):4–11, March 1960.

[29] Shigeru Miyamoto and Takashi Tezuka. Super Mario Bros. (Game), 1985.

[30] Afshin Mobramaein, Morteza Behrooz, and Jim Whitehead. CADI—A conversational assistive design interface for discovering pong variants. *Proceedings of the Fourteenth Artificial Intelligence in Interactive Digital Entertainment Conference (AIIDE)*, 2018.

[31] Afshin Mobramaein and Jim Whitehead. A methodology for designing natural language interfaces for procedural content generation. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, Foundations of Digital Games (FDG) '19, pages 102:1–102:9. ACM, 2019.

[32] Afshin Mobramaein, Jim Whitehead, and Chandranil Chakraborttii. Talk to me about pong: On using conversational interfaces for Mixed-Initiative game design. *2018 AAAI Spring Symposium on the User Experience for Artificial Intelligence (UX of AI)*, 2018.

[33] Darius Monsef. Colourlovers.com (API), 2017.

[34] Reiichiro Nakano. Arbitrary style transfer in the browser (software), 2019.

[35] Nicholas Negroponte. Soft Architecture Machines, 1975.

[36] Mark Nelson, Simon Colton, Edward Powley, Swen Gaudl, Peter Ivey, Rob Saunders, Blanca Perez Ferrer, and Michael Cook. Mixed-Initiative approaches to On-Device mobile game design. In *"Proceedings of the Mixed Initiative Creative Interfaces workshop at CHI"*, 2016.

[37] Mark Nelson, Swen Gaudl, Simon Colton, Edward Powley, Blanca Perez Ferrer, Rob Saunders, Peter Ivey, and Michael Cook. Fluidic games in cultural contexts. In *International Conference on Computational Creativity 2017*, 2017.

[38] Mark J Nelson, Swen E Gaudl, Simon Colton, and Sebastian Deterding. Curious users of casual creators. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*, Foundations of Digital Games (FDG) '18, pages 61:1–61:6, New York, NY, USA, 2018. ACM.

[39] Mark J Nelson and Michael Mateas. Towards automated game design. In *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing*, pages 626–637. Springer Berlin Heidelberg, 2007.

[40] Mark J Nelson and Michael Mateas. An interactive game-design assistant. In *Proceedings of the 13th International Conference on Intelligent User Interfaces*, IUI '08, pages 90–98, New York, NY, USA, 2008. ACM.

[41] Dong Nguyen. Flappy bird (game), 2013.

[42] University of Washington Center for Game Science. Refraction (game), 2010.

[43] Barney Pell. METAGAME: A new challenge for games and learning. In *Heuristic Programming in Artificial Intelligence 3 – The Third Computer Olympiad.*, 1992.

[44] Christian Petry. Online texture generator (software), 2019.

[45] Edward J Powley, Mark J Nelson, Swen E Gaudl, Simon Colton, Blanca Pérez Ferrer, Rob Saunders, Peter Ivey, and Michael Cook. Wevva: Democratising game design. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2017.

[46] David Price, Ellen Rilofff, Joseph Zachary, and Brandon Harvey. NaturalJava: A natural language interface for programming in java. In *Proceedings of the 5th International Conference on Intelligent User Interfaces*, IUI '00, pages 207–211, New York, NY, USA, 2000. ACM.

[47] James Owen Ryan, Eric Kaltman, Timothy Hong, Katherine Isbister, Michael Mateas, and Noah Wardrip-Fruin. GameNet and GameSage: Videogame discovery as design insight. In *Digital Games Research Association (DiGRA)/Foundation of Digital Games (FDG)*, 2016.

[48] Jimmy Secretan, Nicholas Beato, David B D Ambrosio, Adelein Rodriguez, Adam Campbell, and Kenneth O Stanley. Picbreeder: Evolving pictures collaboratively online. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 1759–1768, New York, NY, USA, 2008. ACM.

[49] Vidya Setlur, Sarah E. Battersby, Melanie Tory, Rich Gossweiler, and Angel X. Chang. Eviza: A natural language interface for visual analysis. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, pages 365–377, New York, NY, USA, 2016. ACM.

[50] Noor Shaker, Mohammad Shaker, and Julian Togelius. Ropossum: An authoring tool for designing, optimizing and solving cut the rope levels. In *Proceedings of the 9th Artificial Intelligence in Interactive Digital Entertainment Conference*, 2013.

[51] Noor Shaker, Julian Togelius, and Mark J Nelson. *Procedural Content Generation in Games*. Springer, Cham, 2016.

[52] A M Smith and M Mateas. Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 273–280, August 2010.

[53] Gillian Smith, Jim Whitehead, and Michael Mateas. Tanagra: A mixed-initiative level design tool. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, Foundations of Digital Games (FDG) '10, pages 209–216, New York, NY, USA, 2010. ACM.

[54] Steve Swink. *Game Feel :A game designer's guide to virtual sensation*. Morgan Kaufman Publishers, 2008.

[55] Ko Takeuchi, Goro Abe, and Ryutaro Takahashi. Warioware, inc.: Mega microgames!, 2003.

[56] Unity Technologies. Water4 (Unity3D plugin), 2017.

[57] Unity Technologies. Unity3d (game engine), 2018.

[58] J Togelius and J Schmidhuber. An experiment in automatic game design. In *2008 IEEE Symposium On Computational Intelligence and Games*, pages 111–118, December 2008.

[59] Michael Toy, Glenn Wichmann, Ken Arnold, and Jon Lane. Rogue (game), 1977.

[60] Mike Treanor, Bryan Blackford, Michael Mateas, and Ian Bogost. Game-O-Matic: Generating videogames that represent ideas. In *Proceedings of the The third workshop on Procedural Content Generation in Games*, page 11. ACM, May 2012.

[61] Firedrop Ventures. Sacha (AI software), 2018.

[62] Bruce Wilcox. Chatscript (NLU language), 2010.

[63] ZeptoLab. Cut the rope (game), 2010.

[64] J Zhu, A Liapis, S Risi, R Bidarra, and G M Youngblood. Explainable AI for designers: A Human-Centered perspective on Mixed-Initiative Co-Creation. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, August 2018.